To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

应用笔记

NEC 微控制器的安全装置

文档编号：U18270CA1V0AN00(第一版)
发行日期：2008 年 3 月
© NEC Electronics Corporation 2007
日本印制

目录:

1 NEC 微控制器的安全装置 6
1.1 安全装置的概述 6
1.1.1 复位 6
1.1.2 上电清零(POC)复位 7
1.1.3 低电压检测 (LVI) 8
1.1.4 看门狗定时器(WDT) 9
1.2 程序描述和规范 11
1.3 软件流程图 13
1.3.1 程序启动和初始化 13
1.3.2 WDT_Init() – 看门狗定时器初始化，选项字节 0080H 和 0081H 的设定 14
1.3.3 TM00_Init() – 用于 1 毫秒周期中断的定时器 00 初始化 15
1.3.4 TMH1_Init() –用于 10 毫秒周期中断的定时器 H1 初始化 16
1.3.5 LVI_Init() – 低电压检测初始化 17
1.3.6 Main() – 主程序 – 安全装置示范 19
1.3.7 LVI_Reset_239() –设定在 VDD 低于 2.39V 时用于复位的 LVI 21
1.3.8 LVI_Reset_393() –设定在 VDD 低于 3.93V 时用于复位的 LVI 22
1.3.9 LVI_Int_239() – 设定在 VDD 低于 2.39V 时用于中断的 LVI 23
1.3.10 LVI_Int_393() – 设定在 VDD 低于 3.93V 时用于中断的 LVI 24
1.3.11 EXLVI_Reset() – 设定在 EXLVI 低电压时用于复位的 LVI 25
1.3.12 EXLVI_Int() – 设定在 EXLVI 低电压时用于中断的 LVI 26
1.3.13 WDT_Demo() – 示范看门狗定时器溢出 27
1.3.14 MD_INTTMH1 – 定时器 H1 中断服务程序 28
1.3.15 MD_INTTM000 – 定时器 00 中断服务程序 28
1.3.16 MD_INTLVI – LVI 中断服务程序 29
1.4 Applilet 参考驱动器 30
1.4.1 用于看门狗定时器(WDT)的 Applilet 的配置 30
1.4.2 用于上电清零(POC)的 Applilet 的配置 30
1.4.3 用于低电压检测其(LVI)的 Applilet 的配置 30
1.4.4 用于产生 1 毫秒间隔中断的定时器 TM00 的 Applilet 的配置 31
1.4.5 用于产生 10 毫秒间隔中断的定时器 TMH1 的 Applilet 的配置 32
1.4.6 Applilet 生成代码，选择选项功能 33
1.4.7 用于看门狗定时器(WDT)和 POC 的 Applilet 生成文件和函数 34
1.4.8 用于低电压检测器(WDT)的 Applilet 生成文件和函数 35
1.4.9 用于 TM00 和 TMH1 的 Applilet 生成文件和函数 37
1.4.10 用于端口初始化的 Applilet 生成文件和函数 38
1.4.11 其它 Applilet 生成文件 39
1.4.12 不由 Applilet 生成的示范程序文件 39
1.5 示范平台 40
1.5.1 源 40
1.5.2 程序示范 40
1.5.3 使用 M-Station VDD 选择替代电位器的示范 43
1.6 硬件框图 44
1.7 软件模块 45
2 附录 A – 开发工具 46
2.1 软件工具 46
2.2 硬件工具 46
3 附录 B – 软件列表 47
3.1 Main.c 47

3.2 Macrodriver.h 51
3.3 System.h 52
3.4 Systeminit.c 53
3.5 System.c 55
3.6 Lvi.h 56
3.7 Lvi.c 56
3.8 Lvi_user.c 58
3.9 Port.h 60
3.10 Port.c 61
3.11 Timer.h 62
3.12 Timer.c 64
3.13 Timer_user.c 67
3.14 Watchdogtimer.h 68
3.15 Watchdogtimer.c 69
3.16 Led_0537.h 71
3.17 Led_0537.c 72
3.18 Sw_0537.h 74
3.19 Sw_0537.c 74
3.20 Option.inc 76
3.21 Option.asm 77
3.22 Option_POC27.inc 78
3.23 Option_POC27.asm 79

引言

本文档的目的是为包括 NEC 微控制器设备在内的外围提供简单的例子。用户可以通过这个来增强对这些外围使用的理解。

文档的格式包括以下:

- 外围结构的描述
- 例子程序的描述和规范
- 软件流程图
- Applilet 参考驱动
- 使用的平台示范
- 硬件框图
- 软件模块

Applilet 是一个为外围生成驱动代码软件工具。它为片上外围方便迅速的生成代码。

用户应当参考设备用户手册和其它详细相关文档。

安全装置的概述

为 NEC 微控制器安装的安全装置可以在开始操作时复位系统或者从复位条件重启。当电压下降到指定电平时，低电压检测电路检测并中断系统，以保持系统状态的安全。如果程序进入了死循环，看门狗定时器将复位系统并且从头开始。这些功能都是保证系统安全运行条件的重要部分。

本应用笔记将展示如何使用安全装置的例子:

- 通过外部 RESET 引脚复位
- 上电清零复位
- 低电压检测器复位或者中断
- 看门狗定时器复位

1 NEC微控制器的安全装置

安装在 NEC 微控制内的安全装置如下:

- 复位
- 上电清零复位
- 低电压检测
- 看门狗定时器

1.1 安全装置的概述

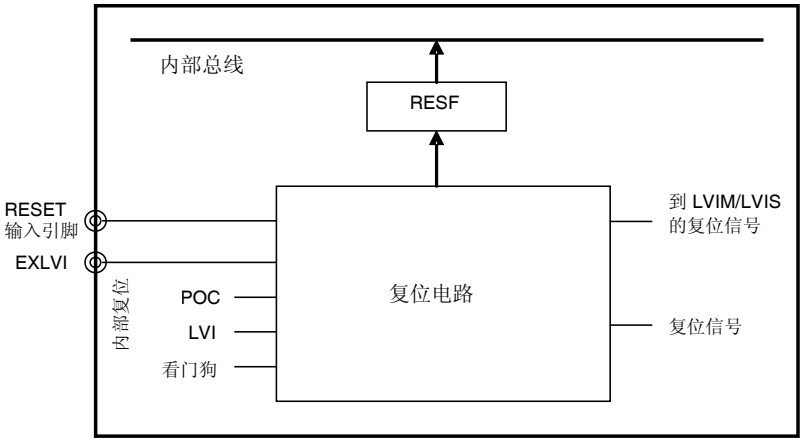
这部分提供一个每个安全装置简单的概述和使用这些装置的示范。详细的内容，将描述 NEC 78K0/Kx2 系列设备中的安全装置。

1.1.1 复位

下列硬件资源产生复位信号。

- 外部 RESET 输入引脚
- 通过看门狗定时期溢出来内部复位
- 通过上电复位来内部复位
- 通过低电压检测来内部复位

外部和内部复位没有功能区别。这两种情况下，当产生一个复位信号，程序通过复位向量表(0000H-0001H)在指定地址开始执行。当产生一个复位信号，I/O 寄存器和端口引脚 (RESF 除外)设定为缺省值。

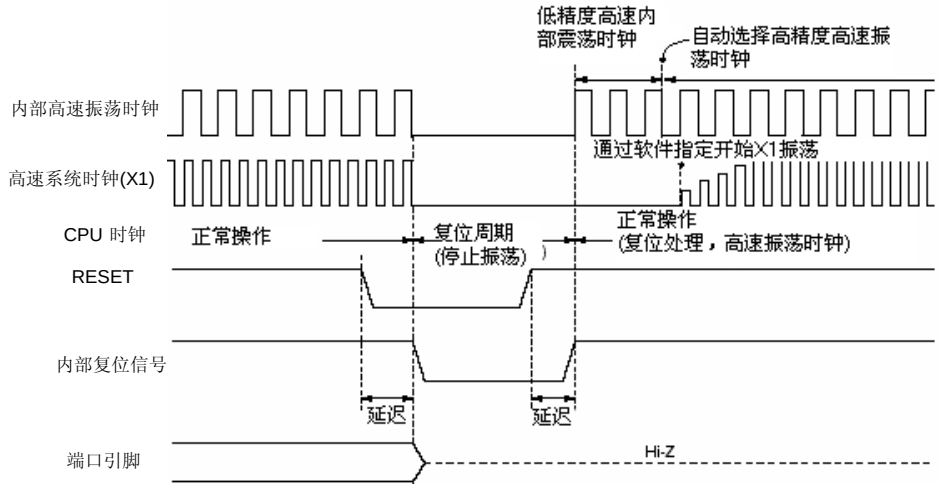


设定复位标志和并且可以识别复位的种类。

当产生复位请求时的 RESF 状态

复位源 / 标志	RESET 输入	PCC复位	WDT复位	LVI复位
WDTRF	清除 (0)	清除 (0)	设置 (1)	保持
LVIRF			保持	设置 (1)

外部 RESET 引脚和内部复位信号的时序图如下所示。当产生或者释放 RESET 时，有个避免重复内部复位在 RESET 引脚产生噪声的延迟，程序执行状态使用内部高速震荡时钟。外部 X1 振荡器稳定之后，CPU 始终可以切换到使用 X1 作为系统时钟。



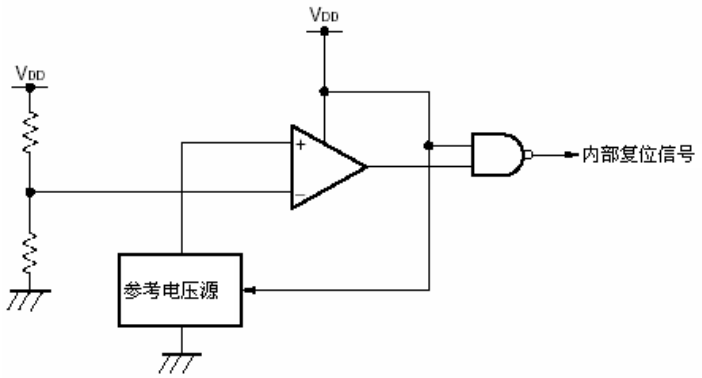
1.1.2 上电清零(POC)复位

上电清零提供以下功能:

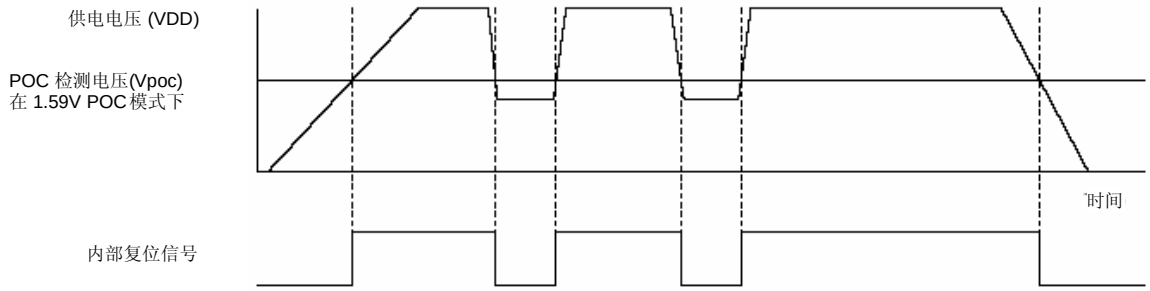
- 当 VDD 电压上电稳定后在复位时保持设备
- 可以选择上电 POC 电压(通过选项字节 1.59V 或 2.7V)
- 如果 VDD 电压下降到可能导致操作不稳定的电平时复位设备

上电清零在 VDD 上电时产生一个内部复位。在 1.59V 模式下，当电源电压超过 VDD = 1.59V 脉冲/分钟 0.5V 时释放内部复位信号。在 2.7V/1.59V 模式下，当电源电压超过 VDD = 2.7V 脉冲/分钟 0.2V 时释放内部复位信号。工作期间，上电清零功能同样检测 VDD 供电电压下降到 1.59V 以下并且产生内部复位信号。

上电清零电路



在上电清零电路中，供电电压(VDD)和检测电压(Vpoc)相比较。当 VDD 下降到检测电压(Vpoc)之下时产生内部复位信号。当 VDD 上升到检测电压(Vpoc)之上时复位释放。



Vpoc 电压可以在上电时选择；这是在 flash 存储器中一个指定区域。对于 78K0/Kx2 系列设备，是 0081H，位 0 (POCMODE)指定上电时的 Vpoc 电压。如果 POCMODE 是 0，Vpoc 电压是 1.59V；如果 POCMODE 是 1，Vpoc 上电电压是 2.7V。

两种情况中，Vpoc 电压上电后都是 1.59V；如果 VDD 降低到这个电平之下，产生一个内部复位。在电压再次回升到 1.59V 之前设备将被保持。

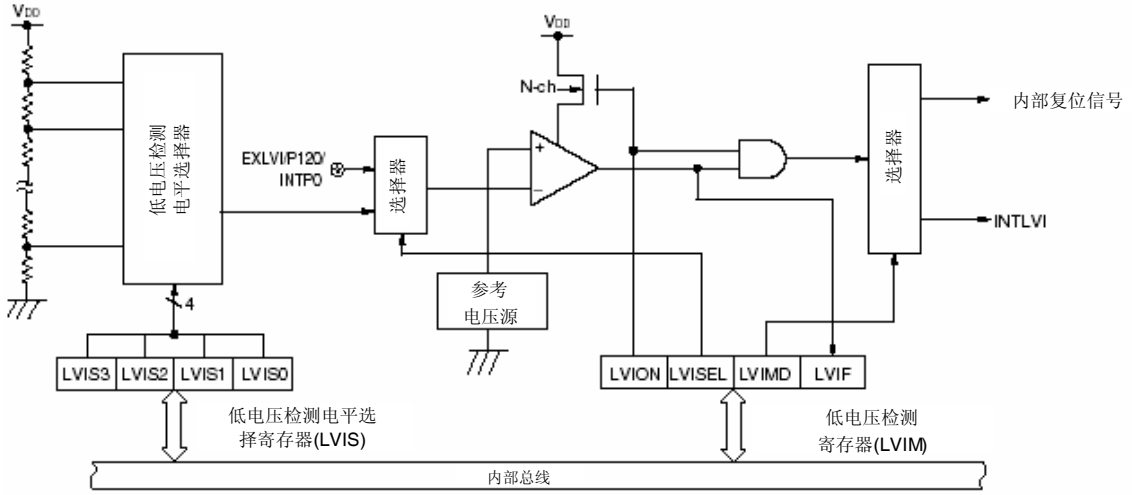
1.1.3 低电压检测 (LVI)

低电压检测 (LVI)提供如下功能:

- 选择用于低电压条件下的复位或者中断操作
- 检测 VDD 电压，或者外部输入引脚
- 对于 VDD 电压，可以设定十六分之一电压

低电压检测电路把电压和内部参考进行比较，并且当电压低于设定地电压范围时产生一个内部中断信号或者中断复位信号。比较电压可能是 VDD 供电电源，并且一个可以检测的电阻分压器的漏电流可以设定为选择电压电平。低电压检测电路同样支持外部输入，EXLVI，并且可以当这个电压下降到参考电压 1.21V 之下时产生一个中断或者复位。

低电压检测器框图



控制低电压检测的寄存器是:

寄存器	*符号*	*功能描述*
低电压检测寄存器	LVIM	设定低电压检测和操作模式
		允许/禁止低电压检测
		选择内部或者外部输入 (EXLVI)
		设定低电压操作模式和标志
低电压检测电平选择寄存器	LVIS	设定低电压检测电平
		选择到 16 级别
端口模式寄存器	PMx	设定端口模式来输入或者输出 EXLVI 引脚

要为操作配置低电压检测功能，使用下列步骤:

当使用复位模式:

- 在 MK0L 寄存器中设定 LVIMK 位来屏蔽 LVI 中断
- 清除 IF0L 寄存器中的 LVIIF 中断标志
- 通过清除 LVIM 寄存器的 LVION 位来禁止 LVI 检测器
- 使用 LVIM 寄存器的 LVISEL 位来选择 VDD 检测或 EXLVI 检测模式
- 如果使用 VDD 检测模式，在寄存器 LVIS 中设定电压电平
- 如果使用 EXLVI 检测模式，通过适当的 PM 寄存器设定 EXLVI 引脚来输入
- 设定 LVIMD 位为 0 来临时选择中断模式
- 开启 LVION 位来使检测器有效
- 等待检测器的一个最小的 10 毫秒来稳定
- 设定 LVIMD 位为 1 来选择复位模式

当使用中断模式:

- 在 MK0L 寄存器中设定 LVIMK 位来屏蔽 LVI 中断
- 清除 IF0L 寄存器中的 LVIIF 中断标志
- 通过清除 LVIM 寄存器的 LVION 位来禁止 LVI 检测器
- 使用 LVIM 寄存器的 LVISEL 位来选择 VDD 检测或 EXLVI 检测模式
- 如果使用 VDD 检测模式，在寄存器 LVIS 中设定电压电平
- 如果使用 EXLVI 检测模式，通过适当的 PM 寄存器设定 EXLVI 引脚来输入
- 设定 LVIMD 位为 0 来临时中断模式
- 使用 PR0L 寄存器设定 LVI 中断的优先级
- 开启 LVION 位来使检测器有效
- 等待检测器的一个最小的 10 毫秒来稳定
- 清除 LVIMK 位为 0 来允许 LVI 中断

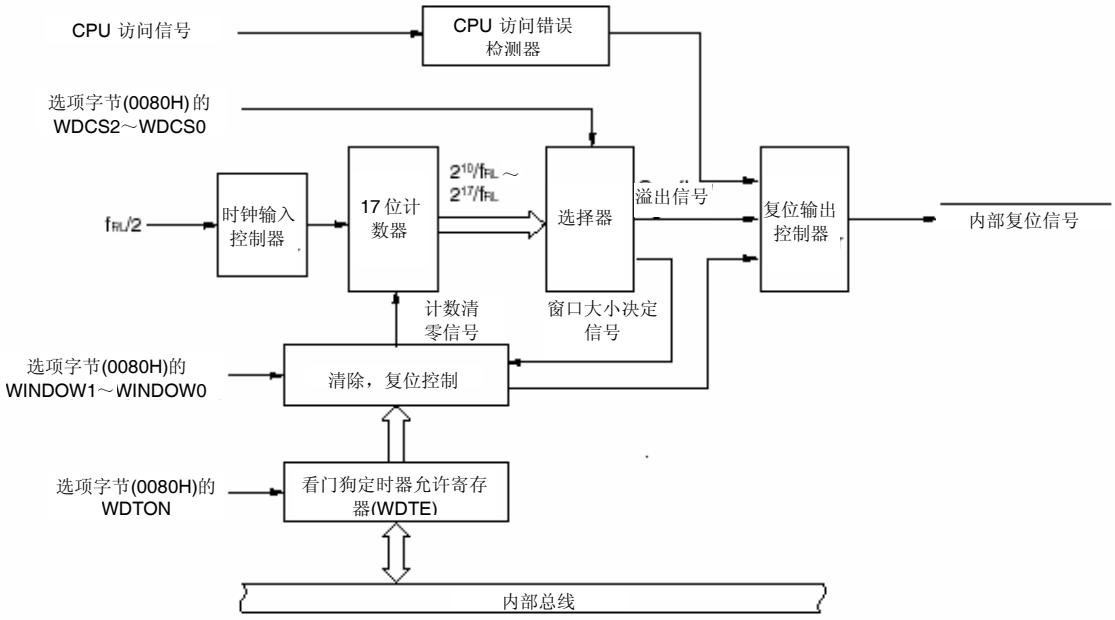
1.1.4 看门狗定时器(WDT)

看门狗定时器 (WDT)提供如下功能:

- 为跑飞循环程序复位处理器
- 基于内部振荡器操作，即使外部时钟失效时仍然工作
- 选择时钟频率来控制定时器必须被清除的频率
- 对于停止内部振荡器的可选保护
- 对于不正确值写入到看门狗定时器的保护
- 对于在不正确时间(开窗功能)清除看门狗定时器的可选保护
- 取指令或者读/写到非法存储器区域时复位

看门狗定时器(WDT)基于内部低速振荡时钟工作。看门狗定时器用来检测无意的程序循环。当允许并且运行，如果看门狗定时器计数器没有定期清除，计数器溢出并且产生一个内部复位信号。

当使用看门狗定时器，用户程序必须通过向看门狗定时器允许寄存器(WDTE)写入一个指定值来清除看门狗定时器计数器。这个在主程序循环中必须定期执行，或者在任何任何一部分执行时间超过看门狗定时器计数器溢出时间的代码中执行。



根据使用特别的 NEC 微控制器，看门狗定时器应该通过写入寄存器开始，或者通过在 flash 存储器中的一个设定值开始。一旦看门狗定时器开始运行，它无法被禁止，预防出现偶尔禁止该寄存器时出现程序跑飞。另外，它可以选择的低速振荡器模式不能通过软件禁止，以防看门狗定时器时钟源被停止。

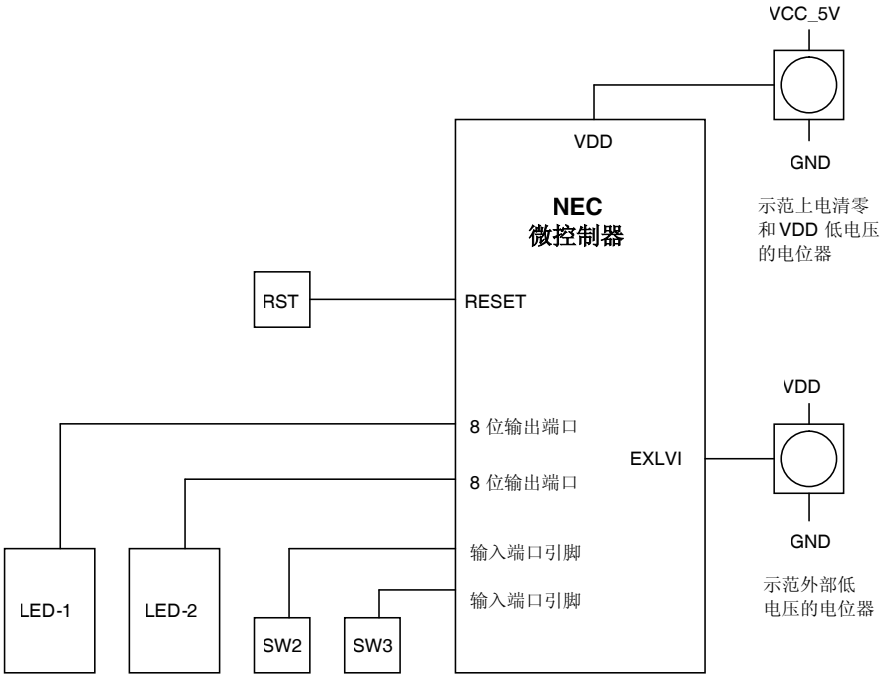
通过写入一个指定值到 WDTE 寄存器来清除看门狗定时器；在 78K0/Kx2 系列中，这个值是 0ACH 。如果另外的值写入到 WDTE，将产生一个复位，保护写入错误值到 WDTE 的程序循环。

看门狗定时器开窗周期保护一个跑飞程序写入正确值(0ACH)到 WDTE 寄存器，但是由于程序故障也会在错误时间这样做。如果开窗周期设定为 25%，看门狗定时器计数器可以仅通过在经过溢出时间的 75%之后写入 0ACH 到 WDTE 寄存器来清除，但是要在溢出时间到达之前。这要求正确的程序结构来确保在正确的时间写入 WDTE 寄存器。开创周期可以设定为 100%，这样就允许任何时间写入 WDTE 寄存器正确的值。

在示范程序中用到的 78K0/Kx2 NEC 微控制器系列中，开始看门狗定时器通过包含在 0080H 选项字节的一个位来设定。这个选项字节同样选择看门狗定时器计数器的频率，和看门狗开窗时间。

1.2 程序描述和规范

用于示范复位，上电清零，低电压检测，和看门狗定时器操作，NEC 微控制器将连接到一个复位切换，其它切换为输入，输出一个两位 LED 显示，和用于 VDD 和 EXLVI 的可变电压控制。通过连接一个连接到设备的 VDD 电源电位器的抽头，使用一个可调整的 VDD 电压。作为转换电位器，抽头的电压将在 0V 和 5V 之间转换，允许检测低电压操作。



启动时，示范程序将读取 RESF 寄存器并且在 LED 上显示值。这允许用户看到上一个复位。

要示范外部复位，RST 切换可能连接 RESET 输入到地。

要示范上电清零，设定电位器使 VDD 电压低于 Vpoc 电压。这时，设备将在复位中保持，并且因为没有驱动 LED 显示将为空。改变电位器使 VDD 电压上升到 Vpoc 电压之上,处理器将从复位状态出来，将显示 RESF 标志值；上电清零复位将为“00”。减低电压 VDD 低于 1.59V 将导致上电清零条件，LED 显示将为空。增加 VDD 电压回到 1.59V 之上将释放上电清零复位，将再次显示 RESF 的值“00”。

要示范低电压检测 (LVI)操作，使用相同的电位器。RESF 的值显示后，用户可以通过按 SW2 选择测试中的显示 LVI 操作，按 SW3 执行它。要示范 VDD 检测，设定程序两个 VDD 电压电平中的一个,并且上升用于复位或者重点操作的 LVI 单位。降低 VDD 电压到 VDD 设定点将导致复位(显示空)或者一个中断(显示“LI”)。当电压上升，清除复位，并且 RESF 标志将显示 LVI 作为复位源。

要示范 EXLVI，连接一个 VDD 和 GND 的分离电位器到 EXLVI 引脚，选择一个用于复位或者中断模式的 EXLVI，然后降低 EXLVI 输入低于 1.21V EXLVI 临界值，产生一个复位(显示空)或者一个中断(显示“LI”)，上升 EXLVI 电压将清除复位，并且显示 RESF 标志 LVI 作为复位源。

要示范看门狗定时器，程序将要设定一个用于周期中断定时器在看门狗定时器溢出之前来清除它。用户可以选择看门狗定时器示范测试；将有一个闪烁值；闪烁的速度和显示值将由中断频率决定。

重复按 SW3 将增加定时器中断间隔，将在显示中增加值，和一个慢速闪烁率。当间隔超过看门狗定时器溢出时间，程序将被复位，并且显示 RESF 值标志一个看门狗定时器复位。

规范:

- 将设置 POC 电压为 1.59V 模式 (可以选择使用 2.7V/1.59V 模式)
- 最初将为 VDD 低于 3.93V 中断设置 LVI
- 将显示 LVI 复位和 LVI 中断操作
- 将设置 LVI VDD 电压为 3.93V 和 2.39V
- 将为 68 毫秒溢出时间设置看门狗定时器，100%开窗周期
- 将为清除看门狗定时器使用可变间隔的定时器 H1 作为周期性中断
- 将通过使用定时器 00 考虑 10 毫秒稳定来去抖动输入切换。

1.3 软件流程图

示范程序由下列主要部分组成:

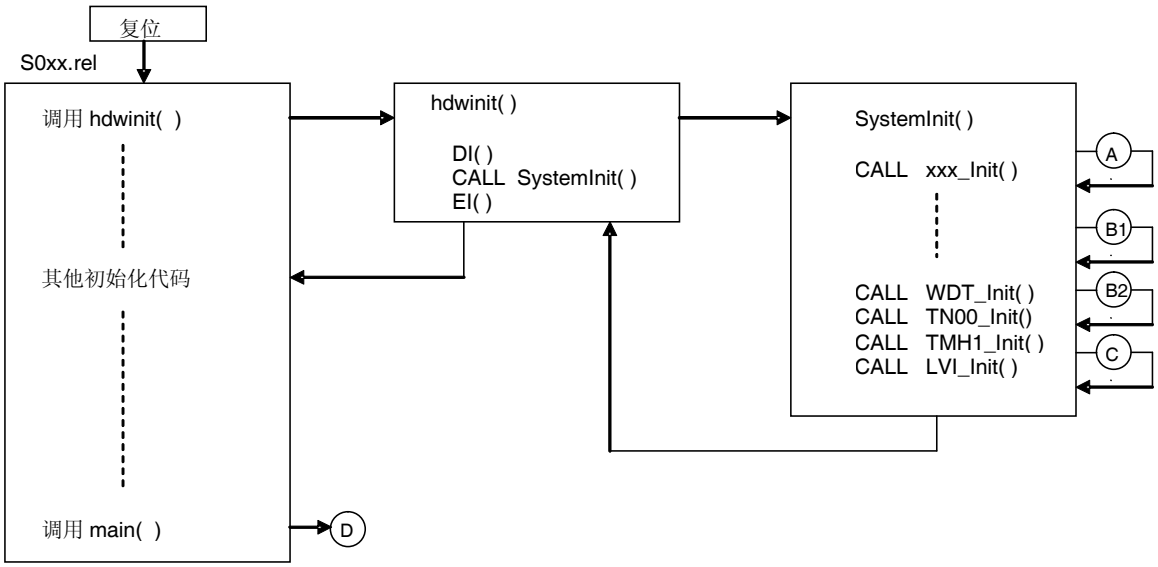
- 初始化程序的代码，在 main()程序开始之前调用;
- 主程序循环，显示复位标志，检查切换和运行测试状态
- LVI，EXLVI，和看门狗定时器 (WDT)功能示范中的子程序
- 由 Applilet 生成的操作定时器 00 和定时器 H1 开始，停止和改变的子程序
- 由 Applilet 生成的开始，停止和改变 LVI 条件的子程序
- 由 Applilet 生成的用于清除看门狗定时器的子程序
- 具有用于操作定时器和 LVI 中断用户码(Applilet 产生的基础中断服务程序，附加了用户码)的子程序
- 用于切换输入和 LED 显示输出的子程序

这里流程图将描述初始化，除程序，示范子程序，和定时器以及 LVI 中断服务程序。这里这个流程图里面没有其他程序，但是将在。1.4 节中描述，并且在附录 A 中包括全部列表。

1.3.1 程序启动和初始化

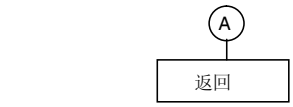
C 语言写的 78K0 程序中，通过 C 语言提供的目标代码文件起始代码例如 s0l.rel，它是连接到用户程序中的。这个起始代码调用一个 hdwinit()函数；用户可以在这里放置任何硬件初始化代码。

当使用 Applilet 产生一个用于 78K0 的 C 程序，hdwinit()函数自动附加到用户程序当中，并且调用函数 SystemInit()。SystemInit()函数为每个硬件依次调用初始化程序。



hdwinit()函数结束后，起始代码调用用户程序的 main()函数。所以在 main()函数的开始，应当执行所有外围的初始化。main()函数不需要调用个别外围初始化程序。

1.3.2 WDT_Init() – 看门狗定时器初始化，选项字节 0080H 和 0081H 的设定



选项字节 (0080H) = 0x78
OPT.7 = 0
OPT.6-5 = 11 (Window1 -0) 100%开窗
OPT.4 = 1 (WDTON)复位后允许
OPT.3-1 = 100 (WDCS20) 68.26 毫秒溢出
OPT.0 = 0 (LSROSC)不能通过软件停止

POC81 字节(0081H) = 0x00
POC81.0 = 1(POCMODE)2.7V/1.59V POC 模式

通过 SystemInit()调用 WDT_Init()函数作为硬件初始化步骤的一部分。这个程序只包括一个返回值。

在用于示范程序的 NEC 微控制器中，用于看门狗定时器的初始化设定和启动由选项字节来控制，位于 flash 存储器的 0080H。在 NEC 其它的微控制其中，看门狗定时器的设定和开始可能由寄存器控制，这种情况下代码应当放在 WDT_Init()程序中。

选项字节 0080H 的位 4，WDTON，控制看门狗定时器是否在复位释放后运行。在我们的示范程序中，这位设为 1，看门狗定时器运行。

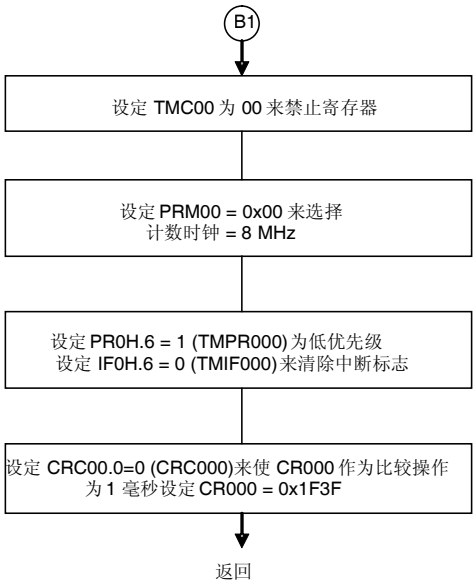
选项字节 0080H 的位 6 和 5， WINDOW1 和 WINDOW0，控制看门狗定时器的开窗时间。我们设置这两位为 1，开窗时间为 100%。这时不限制什么时候写入看门狗定时器来清除它，重新设定溢出定时器。

选项字节 0080H 的位 3，2，和 1，WDCS2～WDCS0，控制看门狗定时器溢出时间。这些为选择不同数目的用于溢出低速振荡器的计数。这些值由 3 位来设定，100，选择 16,384/f_{RL} ；低速振荡器的频率是 240 KHz，看门狗定时器溢出时间是 16,384/240,000 秒，或者 68.27 毫秒。复位释放后，程序必须每 68.27 毫秒内写入 0xAC 到看门狗定时器寄存器 WDTE。如果超过这个时间都没有写入 0xAC 到 WDTE，处理器将复位。

选项字节 0080H 的位 0，LSROSC，控制低速振荡器是否可以通过软件停止。如果这位是 0，可以通过设定 RCM 寄存器的 LSRSTOP 位来停止低速振荡器；这将停止看门狗定时器记的计数。这个设定同样会停止看门狗定时器在处理器处于 HALT 或者 STOP 模式下的计数，而不管低速振荡器是不是停止。如果 LSROSC 位在选项字节中设为 1，即使设置了 LSRSTOP 位低速振荡器不会被停止。这个设定保证了看门狗定时器一直在运行。在我们的示范程序中，LSROSC 设为 0，允许软件停止低速振荡器。

选项字节 POC81，位置是 0081H，控制上电清零动作。位 0，POCMODE，控制是否选择上电清零操作。如果这位是 0，选择 1.59V 模式，并且当 VDD 电压到达 1.59V 时内部复位释放。如果这位是 1，选择 2.7V/1.59V 模式。在这个模式下，初始上电时当 VDD 到达 2.7V 时内部复位才会释放。两种模式下，当 VDD 低于 1.59V 产生一个内部复位，并且当 VDD 再次回升到这个电平时复位。在我们的示范程序中，我们设定 POCMODE 为来选择 1.59V 模式。

1.3.3 TM00_Init() – 用于 1 毫秒周期中断的定时器 00 初始化



TM00_Init()程序，通过 SystemInit()调用，设定 16 位定时器 00 (TM00)为间隔定时器模式，并且每 1 毫秒产生一个中断。这个中断用于去抖动。

推荐在改变寄存器设定时禁止这个寄存器。

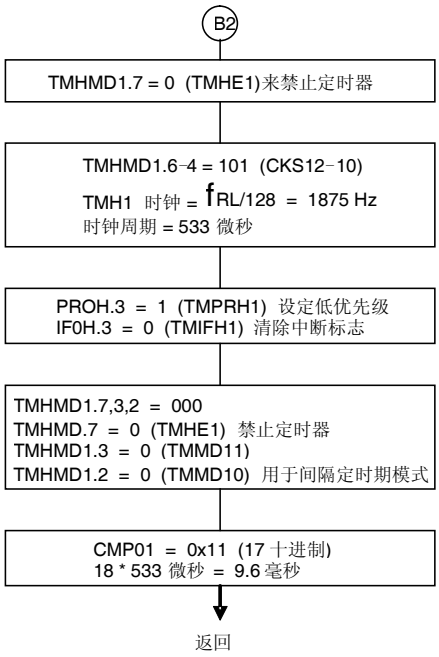
预分频器模式寄存器，PRM00，控制定时器时钟，设为 0x00。这样设定下，定时器 00 将使用 f_{PRS}，主外围时钟，用作它的计数时钟。在我们的示范程序中， f_{PRS} 用于内部 8MHz 高速振荡器作为它的源。这意味着计数寄存器 TM00 将每 1/8,000,000 秒累加一次，也就是 0.125 微秒累加一次。

涉及到 INTTM000 中断的寄存器选择设定为低优先级，清除中断标志。在缺省禁止状态下屏蔽寄存器控制定时器中断的允许将被忽略。

在 NEC 微控制器中，每个中断可能都被赋予高优先级，或者缺省低优先级。在这些相同优先级组里，每个中断源有详细的优先级；在相同的优先级组中，INTTM000 比 INTTMH1 (定时器 H1 中断)或 INTLVI (低电压中断)优先级低。如果这些中断同时产生，INTTM000 将在 INTTMH1 或 INTLVI 之后响应。

CRC00 寄存器设定 CR000 用作为比较寄存器，并且设定比较值为 0x1F3F (7999 十进制)，这个值将使 CR000 和 每个 TM00(7999 + 1)，或者 8,000 个计数相等。这将每 8000*0.125 微秒= 1,000 微秒= 1 毫秒产生一次。这个值 0x1F3F 是通过 Applilet 在详细对话框中计算出来的 1 毫秒指定间隔。

1.3.4 TMH1_Init() –用于 10 毫秒周期中断的定时器 H1 初始化



TMH1_Init()程序，通过 SystemInit()调用，设定 8 位定时器 H1(TMH1)准备为 10 毫秒间隔定时器操作。这个定时器用于产生周期中断来在看门狗定时器溢出之前复位它。

当设定其他内容时，首先通过设定 TMHMD1 寄存器中的 TMHE1 允许位为零来禁止这个定时器。

TMHMD1 的位 6，5，和 4，CKS12，CKS11，和 CKS10 位，为 TMH1 控制时钟源。当这些为设为 101，选择 $f_{RL}/128$；f_{RL} 是低速振荡器，工作在 KHz。结果是 240,000/128 或 1875 Hz 作为 TMH1 的时钟频率；每个计数大概 533 微秒。

低速振荡器是看门狗定时器和 TMH1 的源，所以低速振荡器频率的变化会影响这两个定时器的相关速率。

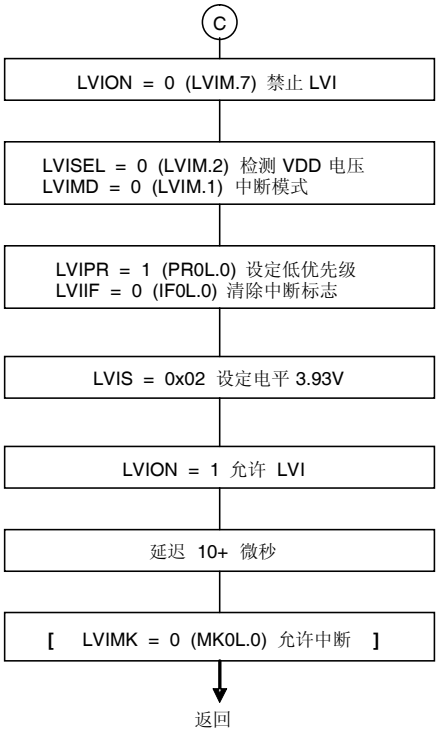
设定 TMPRH1 位(PR0L.3)为 1 来为 INTTMH1 中断选择低优先级，并且清除 TMIFH1 中断标志(IF0L.3)。

在 NEC 微控制器中，每个中断可能都被赋予高优先级，或者缺省低优先级。在这些相同优先级组里，每个中断源有详细的优先级；INTTMH1 比 INTTM000 (定时器 00 中断)优先级高，但是比 INTLVI (低电压中断)优先级低。 这个示范使用全部中断低优先级，保持每个优先级的标准顺序。如果这些中断同时产生，INTTMH1 将在 INTTM000 之前，INTLVI 之后响应。

TMHMD1 的 3 和 2， TMMD11 和 TMMD10 位，控制定时器操作模式。这些设置为 00 来选择间隔定时期操作。这将导致在 TMH1 定时器寄存器何 CMP01 比较寄存器相等时产生 INTTMH1 中断。

CMP01 比较寄存器设置为 0x11 (17 十进制)。这将导致每(17 + 1) * 533 微秒，或者 9.6 毫秒产生一个 INTTMH1 中断。TMH1 定时器停止时 TMHE1 允许位将被忽略；它将通过 TMH1_Start()程序开始。

1.3.5 LVI_Init() – 低电压检测初始化



LVI_Init()函数，通过 SystemInit()调用，设定低电压检测电路的操作模式。初始设置如果 VDD 下降到 3.93V 则产生一个中断。

当设定其他内容时，首先清除 LVION 位(LVIM.7)为零，来禁止 LVI 操作。

LVISEL (LVIM.2)控制电压比较源。它清为零，来选择比较指定 VDD 电阻分压器的抽头和参考电压。抽头通过 LVIS 寄存器选择；这样 LVI 检测器在 VDD 下降到指定电压时触发。如果设置 LVISEL 位为 1，EXLVI 引脚电压和 1.21 参考电压比较，并且 EXLVI 低于 1.21V 时触发检测器。在示范程序中改变 LVISEL 来显示 VDD 触发和 EXLVI 触发。

LVIMD (LVIM.1)控制 LVI 操作模式。它清除为零，来选择中断操作；这时如果 LVI 检测器触发，它将产生 INTLVI 中断。如果设置 LVIMD 位为 1，处理器将在 LVI 触发产生时复位。在示范程序中改变 LVIMD 的设定来显示 LVI 检测器的中断和复位模式。

LVIPR 位(PR0L.0)设为 1 来选择低优先级，并且清除 LVIIF 重点标志(IF0L.0)。

在 NEC 微控制器中，每个中断可能都被赋予高优先级，或者缺省低优先级。在这些相同优先级组里，每个中断源有详细的优先级。 INTLVI 比同组的其他中断优先级都高。这个示范使用全部中断低优先级，保持每个优先级的标准顺序，所以如果这些中断同时产生时 INTLVI 比其他中断优先相应。

如果在它的中断服务程序中一个中断产生并且响应，清除屏蔽中断允许位 IE (PSW..7)。为了能在当前中断程序中响应其他中断服务，IE 位应当在中断服务程序中设置。即使在中断服务程序中设置了 IE 位，比当前中断服务低的优先级中断组将不会被响应，直到当前中断服务程序返回为止。

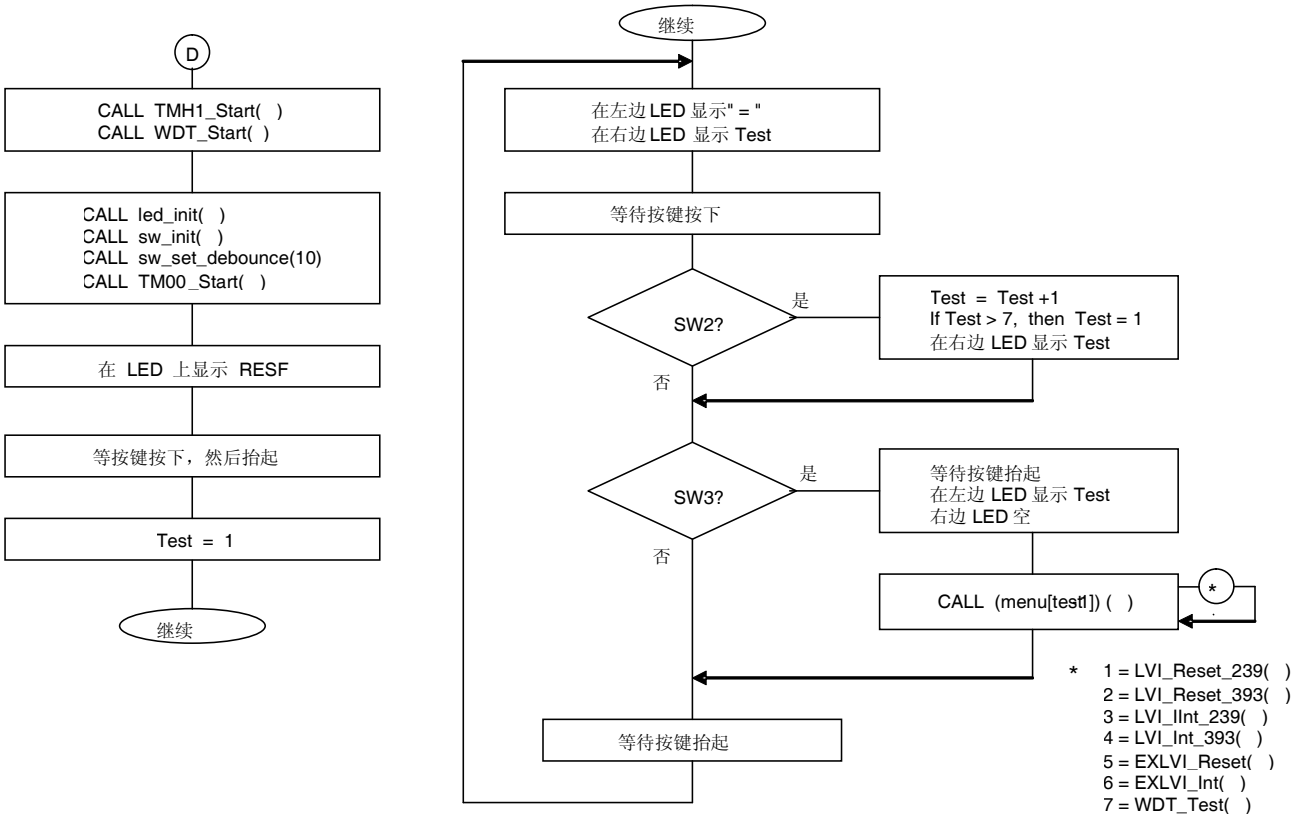
在示范程序中，用于其他中的中断服务程序，用于定时器 00 的 INTTM000 和用于定时器 H1 的 INTTMH1，设定 IE 位来允许 LVI 中断在中断服务程序中产生。

LVIS 寄存器设置为 0x02，在 VDD 电阻分压器选择 3.93V 抽头。如果 VDD 下降到这个电压之下 LVI 将触发。

LVION 位(LVIM.7)设为 1 来允许 LVI 操作。因为出于对最大 10 微秒的稳定的考虑，设定 LVION 之后最小延迟 10 微秒。注意如果这个延迟超过 69 毫秒，看门狗定时器将溢出，产生一个复位。注意时序的循环，如果这个延迟可能超过看门狗定时器溢出时间，应当调用一个 WDT_Restart()来替代。

10 微秒延迟之后，正常 LVI_Init()将通过 LVIMK(MK0L.0)清除来允许 LVI 中断。在我们的示范程序中，我们注释掉这个指令，来保证禁止 LVI 中断，直到测试程序中它才被允许。

1.3.6 Main() – 主程序 – 安全装置示范



main()程序通过起始代码在硬件和软件初始化完成之后调用。main()程序首先通过调用 TMH1_Start()开始 TMH1 定时器，然后调用 WDT_Start()来清除看门狗定时器溢出计数器。

这之后，将产生 INTTMH1 周期中断(启动时每 9.6 毫秒)，并且 MD_INTTMH1()中断服务程序将调用 WDT_Restart()来清除看门狗定时器溢出计数器。只要继续以这个频率产生 INTTMH1 中断，看门狗定时器就不会复位程序。

在示范程序中，处理器的看门狗定时器在复位后通过选项字节开始运行，WDT_Start()程序只能写入 0xAC 到 WDTE 寄存器来清除看门狗定时器溢出。WDT_Restart()做同样的动作。在具有看门狗定时器的 NEC 微控制器开时通过一个寄存器控制，WDT_Start()将开始看门狗定时器。

注意看门狗定时器从复位开始运行，第一次清除是在 main()中调用 WDT_Start()。如果程序启动码花费了超过看门狗定时器溢出的时间(大概 68 毫秒)，然后看门狗定时器将在调用 WDT_Restart()之前复位处理器，程序永远不会到达 main()。如果出现这种情况，应当调整启动代码以一定间隔加入复位看门狗定时器指令。尽早的在起始代码中停止低速振荡器(如果通过选项字节允许的话)，然后到达 mian()立刻重启。

设定定期清除看门狗定时器后，程序初始化 LED 显示和切换，设定切换去抖动计数器为 10，并且调用 TM00_Start()来开始 TM00 定时器。这样之后，TM00 定时器将每个毫秒产生一个 INTM000 中断，并且 MD_INTTM000()中断程序将为防抖检测切换。

main()然后读取 RESF 复位标志寄存器，并且在以十六进制数在 LED 上显示值。我们示范中用于处理器的 RESF 寄存器有两位指示产生复位的类型。

如果一个看门狗定时器计数器溢出发生，或者其它操作导致看门狗定时器产生复位(例如写入 WDTE 的值不是 0xAC)，将设置 RESF 位 4(WDTRF)。此时 LED 显示 “10”。

由于选择了 VDD 检测模式后 VDD 电压，或者选择了 EXLVI 检测模式后 EXLVI 电压低于 1.21V 低于设定阀电平，LVI 检测器产生一个复位，将设置 RESF 位 0(LVIRF)。此时 LED 显示 “01”。

如果通过复位引脚过低 (通过复位按键)，或者通过上电清零电路产生复位，上面的位都被清零，RESF 将是 0x00。此时 LED 显示 “00”。

显示 RESF 值之后，main()等待一个按键按下，然后再抬起，RESF 值将保持到用户按键。

一旦一个按键被按下，main()设定 **test** 变量为 1，然后进入一个循环来允许选择不同的测试。左边 LED 显示 “ = ”，右边 LED 显示当前 **test** 的值。启动时显示将为“ =1 ”。main()然后等待一个按键按下，一直循环到有按键按下。 (INTTM000 和 INTTMH1 中断将周期性的产生)。

一旦按下一个去抖动的按键，main()检测是按下哪个按键。如果是按下 SW2，**test** 变量将增加，如果超过 7 则回到 1。按下 SW，将显示 “=1”，“=2”，“=3”，直到“=7”，然后返回到“=1”。

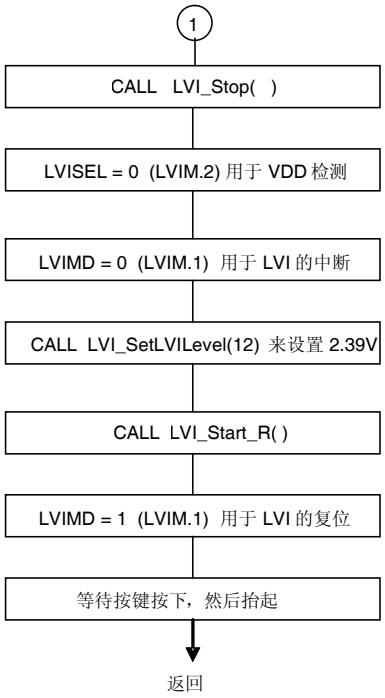
如果是按下 SW3，main()首先等待按键抬起，将 **test** 的值放在左边的 LED，右边的 LED 为空，然后通过从数组 **menu[ ]**获取子程序地址调用一个子程序，使用 **test – 1** 作为矩阵下标。所以如果 **test** 是 1，调用子程序的地址在 **menu[0]**；这是 LVI_Reset_239()程序。

以这样的方法，可以选择并且运行 7 种不同的测试。下面的表格显示 **test** 的值，当 SW3 按下后 LED 的显示，访问菜数组的函数，测试的描述。

Test	**LED**	**menu[test - 1]**	**函数操作**
1	“1 “	LVI_Reset_239()	VDD < 2.39V 时设定 LVI 到 RESET
2	“2 “	LVI_Reset_393()	VDD < 3.93V 时设定 LVI 到 RESET
3	“3 “	LVI_Int_239()	VDD < 2.39V 时设定 LVI 到中断
4	“4 “	LVI_int_393()	VDD < 3.93V 时设定 LVI 到中断
5	“5 “	EXLVI_Reset()	EXLVI < 1.21V 时设定 LVI 到 RESET
6	“6 “	EXLVI_Int()	EXLVI < 1.21V 时设定 LVI 到中断
7	“7 “	WDT_Demo()	WDT 计数器溢出时显示 RESET

当返回选择的测试，main()等待到按键抬起，然后跳转到循环的顶部来重新显示等号和测试号码，等待下一个按键按下。

1.3.7 LVI_Reset_239() –设定在 VDD 低于 2.39V 时用于复位的 LVI



LVI_Reset_239()程序，当选择 Test =1 时从 main()调用，如果 VDD 电压下降到 2.39V 以下，设定 LVI 单位复位处理器。在 LVI_Init()中，或者在其他 LVI 测试程序中这个改变缺省 LVI 性能设定。

调用 LVI_Stop()来清除 LVION 位为 0，禁止 LVI 检测器，并且设定 LVIMK 为 1 来禁止 LVI 中断。

LVISEL 位(LVIM.2)清为零，LVI 检测器的选择 VDD 比较模式。

LVIMD 位(LVIM.1)设为零，临时选择 LVI 检测器的中断模式。

程序调用 LVI_SetLVILevel(12)，设定 LVIS 寄存器为 0x0C。这个设定 VDD 电阻分压器的抽头，如果 VDD 电压下降到 2.39V 之下则 LVI 检测器触发。

程序然后调用 LVI_Start_R()，设定 LVIMK，清除 LVIIF，设定 LVION 为 1 来允许 LVI 检测器，等待至少 10 微秒，然后不清除 LVIMK 位来允许中断返回。这是一个修订版的 LVI_Start()，用于设定复位模式。当返回 LVI_Start_R()，LVI 检测器是稳定的。

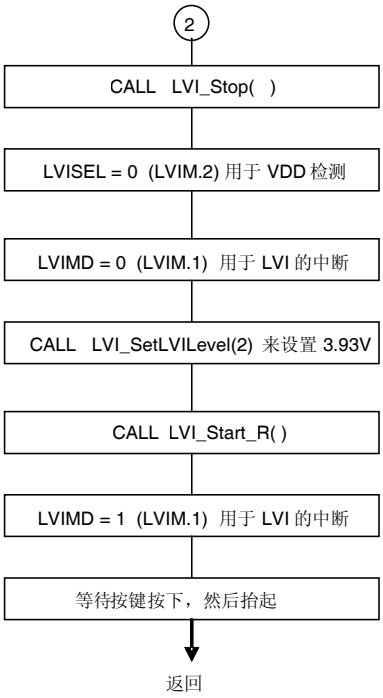
然后 LVIMD 位(LVIM.1)设为 1，选择 LVI 检测器的复位模式。我们选择了 LVI 检测器的复位模式后，LVIMK 的设定将不再影响 LVI 操作。如果 VDD 低于选择电压，会立刻产生一个复位。

如果没有产生复位，是因为 VDD 高于选择电压，在返回到 main()程序之前，测试程序然后等待一个按键被按下和释放。

如果 VDD 高于阀值，当程序等待一个按键时(或者在 main()之后没有设置不同的 LVI 模式)降低它低于阀值，则会产生一个复位。

复位使程序重新启动，重新初始化在 VDD 低于 3.93V 时 LVI 检测器为中断模式。LED 将显示复位标志寄存器的内容“01”，表示该复位是由 LVI 检测器产生。

1.3.8 LVI_Reset_393() –设定在 VDD 低于 3.93V 时用于复位的 LVI



LVI_Reset_393()程序，当选择 Test =2 时从 main()调用，如果 VDD 电压下降到 3.93V 以下，设定 LVI 单位复位处理器。在 LVI_Init()中，或者在其他 LVI 测试程序中这个改变缺省 LVI 性能设定。

调用 LVI_Stop()来清除 LVION 位为 0，禁止 LVI 检测器，并且设定 LVIMK 为 1 来禁止 LVI 中断。

LVISEL 位(LVIM.2)清为零，LVI 检测器的选择 VDD 比较模式。

LVIMD 位(LVIM.1)设为零，临时选择 LVI 检测器的中断模式。

程序调用 LVI_SetLVILevel(12)，设定 LVIS 寄存器为 0x02。这个设定 VDD 电阻分压器的抽头，如果 VDD 电压下降到 3.93V 之下则 LVI 检测器触发。

程序然后调用 LVI_Start_R()，设定 LVIMK，清除 LVIIF，设定 LVION 为 1 来允许 LVI 检测器，等待至少 10 微秒，然后不清除 LVIMK 位来允许中断返回。这是一个修订版的 LVI_Start()，用于设定复位模式。当返回 LVI_Start_R()，LVI 检测器是稳定的。

然后 LVIMD 位(LVIM.1)设为 1，选择 LVI 检测器的复位模式。我们选择了 LVI 检测器的复位模式后，LVIMK 的设定将不再影响 LVI 操作。如果 VDD 低于选择电压，会立刻产生一个复位。

如果没有产生复位，是因为 VDD 高于选择电压，在返回到 main()程序之前，测试程序然后等待一个按键被按下和释放。

如果 VDD 高于阀值，当程序等待一个按键时(或者在 main()之后没有设置不同的 LVI 模式)降低它低于阀值，则会产生一个复位。

复位使程序重新启动，重新初始化在 VDD 低于 3.93V 时 LVI 检测器为中断模式。LED 将显示复位标志寄存器的内容“01”，表示该复位是由 LVI 检测器产生。

1.3.9 LVI_Int_239() – 设定在 VDD 低于 2.39V 时用于中断的 LVI

LVI_Int_239()程序，当选择 Test =3 时从 main()调用，如果 VDD 电压下降到 2.39V 以下，设定 LVI 单位中断处理器。在 LVI_Init()中，或者在其他 LVI 测试程序中这个改变缺省 LVI 性能设定。

调用 LVI_Stop()来清除 LVION 位为 0，禁止 LVI 检测器，并且设定 LVIMK 为 1 来禁止 LVI 中断。

LVISEL 位(LVIM.2)清为零，LVI 检测器的选择 VDD 比较模式。

LVIMD 位(LVIM.1)设为零，选择 LVI 检测器的中断模式。

程序调用 LVI_SetLVILevel(12)，设定 LVIS 寄存器为 0x0C。这个设定 VDD 电阻分压器的抽头，如果 VDD 电压下降到 2.39V 之下则 LVI 检测器触发。

程序然后调用 LVI_Start()，设定 LVIMK，清除 LVIIF，设定 LVION 为 1 来允许 LVI 检测器，等待至少 10 微秒，然后清除 LVIMK 位来允许中断。

如果 VDD 低于选择电压，会立刻在清除 LVIMK 标志之后产生一个 INTLVI 中断。

如果没有产生 INTLVI 中断，是因为 VDD 高于选择电压，在返回到 main()程序之前，测试程序然后等待一个按键被按下和释放。

如果 VDD 高于阀值，当程序等待一个按键时(或者在 main()之后没有设置不同的 LVI 模式)降低它低于阀值，则会产生一个 INTLVI 中断。

在 INTLVI 中， 将执行 MD_INTLVI()中断服务程序，在 LED 上显示“LI”。

1.3.10 LVI_Int_393() – 设定在 VDD 低于 3.93V 时用于中断的 LVI

LVI_Int_393()程序，当选择 Test =4 时从 main()调用，如果 VDD 电压下降到 3.93V 以下，设定 LVI 单位中断处理器。在 LVI_Init()中，或者在其他 LVI 测试程序中这个改变缺省 LVI 性能设定。

调用 LVI_Stop()来清除 LVION 位为 0，禁止 LVI 检测器，并且设定 LVIMK 为 1 来禁止 LVI 中断。

LVISEL 位(LVIM.2)清为零，LVI 检测器的选择 VDD 比较模式。

LVIMD 位(LVIM.1)设为零，选择 LVI 检测器的中断模式。

程序调用 LVI_SetLVILevel(12)，设定 LVIS 寄存器为 0x02。这个设定 VDD 电阻分压器的抽头，如果 VDD 电压下降到 3.93V 之下则 LVI 检测器触发。

程序然后调用 LVI_Start()，设定 LVIMK，清除 LVIIF，设定 LVION 为 1 来允许 LVI 检测器，等待至少 10 微秒，然后清除 LVIMK 位来允许中断。

如果 VDD 低于选择电压，会立刻在清除 LVIMK 标志之后产生一个 INTLVI 中断。

如果没有产生 INTLVI 中断，是因为 VDD 高于选择电压，在返回到 main()程序之前，测试程序然后等待一个按键被按下和释放。

如果 VDD 高于阀值，当程序等待一个按键时(或者在 main()之后没有设置不同的 LVI 模式)降低它低于阀值，则会产生一个 INTLVI 中断。

在 INTLVI 中， 将执行 MD_INTLVI()中断服务程序，在 LED 上显示“LI”。

1.3.11 EXLVI_Reset() – 设定在 EXLVI 低电压时用于复位的 LVI

EXLVI_Reset()程序，当选择 Test =5 时从 main()调用，如果 EXLVI 输入下降到 1.21V 以下，设定 LVI 单位复位处理器。在 LVI_Init()中，或者在其他 LVI 测试程序中这个改变缺省 LVI 性能设定。

调用 LVI_Stop()来清除 LVION 位为 0，禁止 LVI 检测器，并且设定 LVIMK 为 1 来禁止 LVI 中断。

LVISEL 位(LVIM.2)设为 1，LVI 检测器的选择 EXLVI 比较模式。

LVIMD 位(LVIM.1)设为零，临时选择 LVI 检测器的中断模式。

程序然后调用 LVI_Start_R()，设定 LVIMK，清除 LVIIF，设定 LVION 为 1 来允许 LVI 检测器，等待至少 10 微秒，然后清除 LVIMK 位来允许中断。这是一个修订版的 LVI_Start()，用于设定复位模式。当返回 LVI_Start_R()，LVI 检测器是稳定的。

然后 LVIMD 位(LVIM.1)设为 1，选择 LVI 检测器的复位模式。我们选择了 LVI 检测器的复位模式后，LVIMK 的设定将不再影响 LVI 操作。如果 EXLVI 低于选择 1.21V，会立刻产生一个复位。

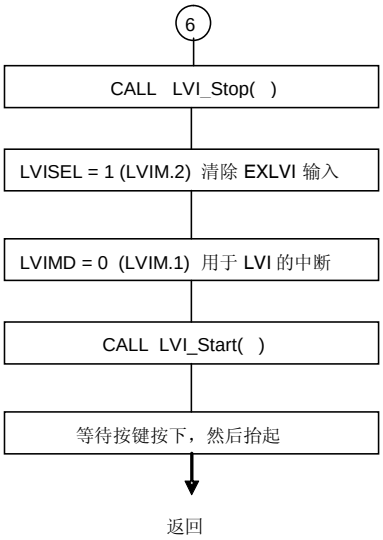
如果没有产生复位，是因为 EXLVI 高于 1.21V，在返回到 main()程序之前，测试程序然后等待一个按键被按下和释放。

如果 EXLVI 高于 1.21V，当程序等待一个按键时(或者在 main()之后没有设置不同的 LVI 模式)降低它低于阀值，则会产生一个复位。

复位使程序重新启动，重新初始化在 VDD 低于 3.93V 时 LVI 检测器为中断模式。LED 将显示复位标志寄存器的内容“01”，表示该复位是由 LVI 检测器产生。

.

1.3.12 EXLVI_Int() – 设定在 EXLVI 低电压时用于中断的 LVI



EXLVI_Int()程序，当选择 Test =6 时从 main()调用，如果 EXLVI 输入下降到 1.21V 以下，设定 LVI 单位中断处理器。在 LVI_Init()中，或者在其他 LVI 测试程序中这个改变缺省 LVI 性能设定。

调用 LVI_Stop()来清除 LVION 位为 0，禁止 LVI 检测器，并且设定 LVIMK 为 1 来禁止 LVI 中断。

LVISEL 位(LVIM.2)设为 1，LVI 检测器的选择 EXLVI 比较模式。

LVIMD 位(LVIM.1)设为零，选择 LVI 检测器的中断模式。

程序然后调用 LVI_Start()，设定 LVIMK，清除 LVIIF，设定 LVION 为 1 来允许 LVI 检测器，等待至少 10 微秒，然后清除 LVIMK 位来允许中断。

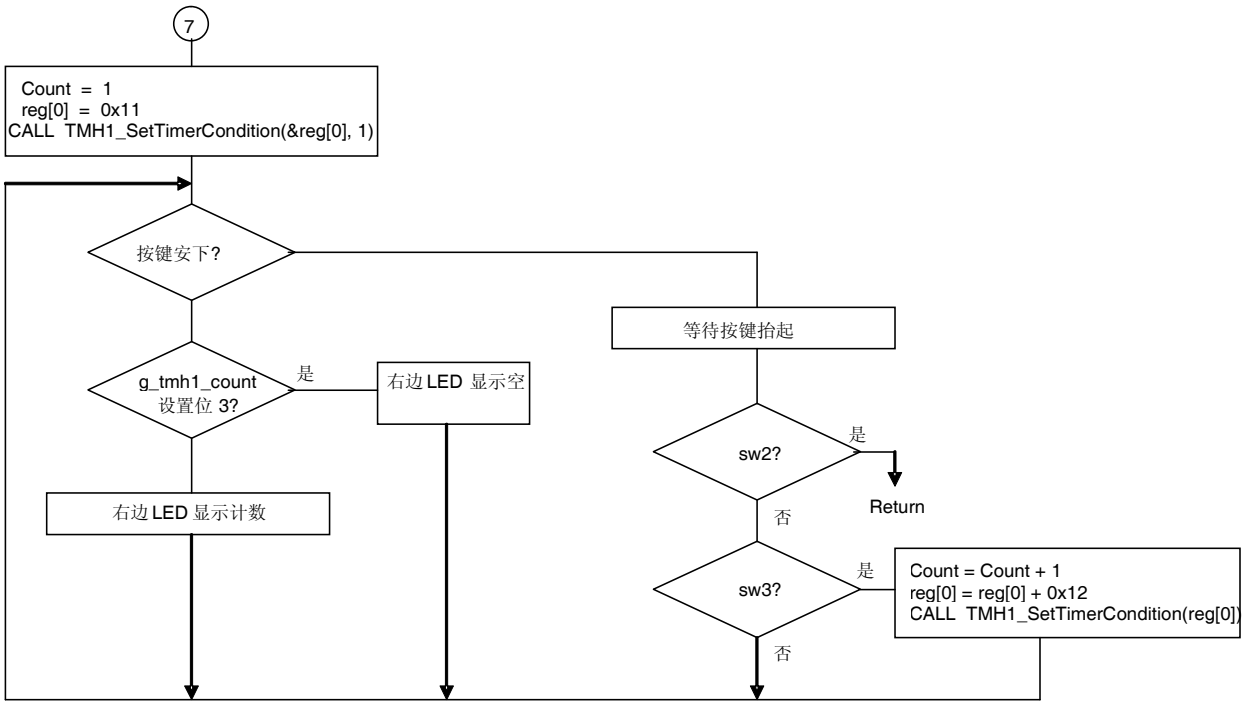
LVIMK 标志清除后，如果 EXLVI 低于选择 1.21V，会立刻产生一个 INTLVI 中断。

如果没有产生 INTLVI 中断，是因为 EXLVI 高于 1.21V，在返回到 main()程序之前，测试程序然后等待一个按键被按下和释放。

如果 EXLVI 高于 1.21V，当程序等待一个按键时(或者在 main()之后没有设置不同的 LVI 模式)降低它低于阀值，则会产生一个 INTLVI 中断。

在 INTLVI 中， 将执行 MD_INTLVI()中断服务程序，在 LED 上显示“LI”。

1.3.13 WDT_Demo() – 示范看门狗定时器溢出



WDT_Demo()程序，当选择 Test =7 时从 main()调用，展示如果溢出前没有清除看门狗定时器会发生什么事情。看门狗定时器溢出时间设为 68 毫秒。

程序开始，**count** 变量设为 1 用于定时器 H1 的间隔设为初始值，0x11，把这个值放在 **reg[0]**变量中并且调用 TMH1_ChangeTimerCondition()。这设定 CMP01 寄存器的值。当 TMH1 寄存器计数到 CMP01 值，产生一个 INTTMH1 中断，MD_INTTMH1()中断服务程序将调用 WDT_Restart()来清除看门狗定时器。当相等时 TMH1 将被清除，并且开始计数下一个 TMH1 时钟。

当 CMP01 设为 0x11 (17 十进制)，TMH1 和 CMP01 相等的间隔是 18 次 TMH1 计数时钟周期。TMH1 的计数时钟是 533.33 微秒，所以 18 * 533.33 是 9.6 毫秒。程序开始，每 9.6 毫秒清除看门狗定时器，这样就不会溢出。MD_INTTMH1()中断服务程序同样会每个 INTTMH1 中断产生时加一全局变量 **g_tmh1_count**；这个变量将在由 TMH1 的间隔决定的频率下计数。

然后 WDT_Demo()程序进入一个循环。等待按键按下；如果没有按键按下，它检测全局变量 **g_tmh1_count**，和依赖位 2 的状态，将都空白右边的数字，或者显示 **count** 变量。**g_tmh1_count** 加一，右边的 LED 数字将以 INTTMH1 中断的一个速度闪烁开和关。

WDT_Demo()程序的开始，左边 LED 通过 main() 设置显示 “7”。所以开始，你将看到“71”， “1”将迅速的闪烁开或关。

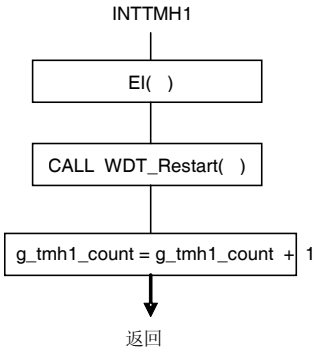
如果按下 SW2，WDT_Demo()返回 main()。如果按下 SW3，加一 **count**， 0x12 (18 十进制)加到 **reg[0]**， 再次调用 TMH1_ChangeTimerCondition()来改变 TMH1 间隔为新值。所以每次按下 SW3，将显示 **count** 为 2, 3, 4, 等等。用于 TMH1 的间隔将改变为 **count** * 18 * 533.33 微秒，或者 **count** * 9.6 毫秒。

间隔增加的值意味着看门狗定时器将每 9.6 毫秒清除一次，然后每 19.2 毫秒，然后每 28.8 毫秒，等等，右边 LED 闪烁将越来越慢。

当 **count** 增加到 8，CMP01 中设定值将是 18 *8 = 144，TMH1 间隔将是 8 * 9.6 毫秒 = 76.8 毫秒。这超出了看门狗定时器溢出时间；所以看门狗定时器将在被 MD_INTTMH1()程序清除前溢出。

处理器将通过看门狗定时器复位，程序将从头开始，显示 RESF 的值。将设置 RESF 位 4 (WDTRF)，只是复位石油看门狗定时器产生，所以将显示 “10”。

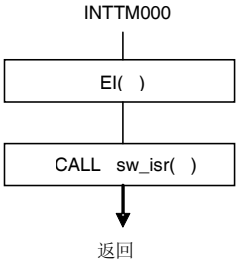
1.3.14 MD_INTTMH1 – 定时器 H1 中断服务程序



如何使用 WDT_Demo()，当定时器计数寄存器 TMH1 和比较寄存器 CMP01 以及 INTTMH1 相等时调用 MD_INTTMH1()中断服务程序。The MD_INTTMH1()程序调用 WDT_Restart()来清除看门狗定时器计数器，并且加一全局变量 **g_tmh1_count**。

注意在这个程序开始时执行 EI()指令；这将允许其他相同优先组合高优先组中断来中断处理器。这将允许在这个中断服务程序中发生 INTLVI 中断。

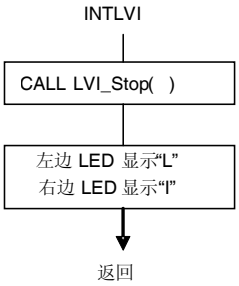
1.3.15 MD_INTTM000 – 定时器 00 中断服务程序



当定时器计数寄存器 TM00 和比较寄存器 CMP00 以及 INTTM000 相等时调用 MD_INTTM000()中断服务程序。这将每 1 毫秒发生一次。The MD_INTTM000()程序调用 sw_isr()来检测按键的抖动。

注意在这个程序开始时执行 EI()指令；这将允许其他相同优先组合高优先组中断来中断处理器。这将允许在这个中断服务程序中发生 INTLVI 或 INTTMH1 中断。

1.3.16 MD_INTLVI – LVI 中断服务程序



当 INTLVI 中断发生时时调用 MD_INTLVI()中断服务程序。当设低电压检测电路为中断模式(LVIMD=0)替代复位模式时，当 VDD 下降到 LVIS 寄存器设定电平(在 VDD 检测模式下)之下时，或者当 EXLVI 下降到 1.21V(在 EXLVI 检测模式下)之下时。

MD_INTLVI()程序调用 LVI_Stop()来禁止 LVI 检测器。用于指示用户产生了一个 INTLVI 中断，LED 显示改变为“LI”。

1.4 Applilet 参考驱动器

NEC 的 Applilet 程序生成器可以自动生成为 NEC 为控制其设备管理外围的 C 或者汇编语言源代码。请参照附录使用的 Applilet 方案。

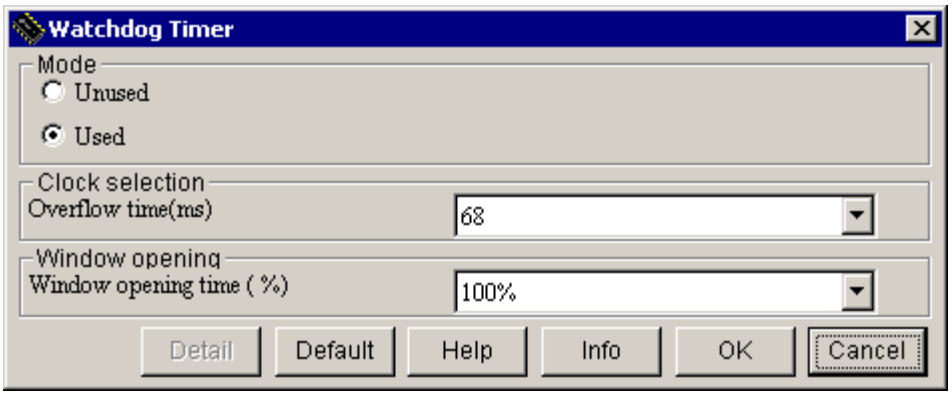
Applilet 用于产生程序的基本初始代码和主函数，看门狗定时器的驱动代码，LVI(低电压)检测，定时器 TM00 和 TMH1，和用于按键输入和 LED 显示输出的 I/O 端口。Applilet 生成基本代码之后，用户定制的程序函数将附加上去。

这部分描述了如何设置 Applilet 来生成用于看门狗定时器，LVI，和定时器的代码以及列出生成的程序。同样列出了附加的文件。

Applilet 开始，建立一个新的工程文件并且保存为.prx 文件。Applilet 显示一个屏幕允许选择不同的外围启动按键。

1.4.1 用于看门狗定时器(WDT)的 Applilet 的配置

选择“Watchdog Timer”按键进入看门狗定时器配置对话框。



选择用于生成看门狗定时器代码的模式。从下拉列表选择看门狗定时器计数器溢出时间为 68 毫秒，和选择开窗时间为 100%。

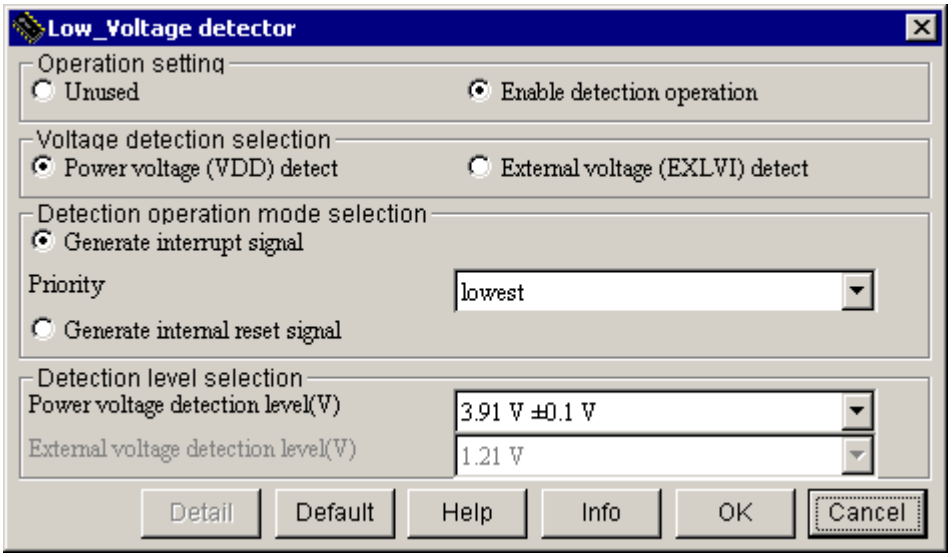
这些选择将使 Applilet 生成适当的值来保存在 0080H 的选项字节中，并且生成程序来控制看门狗定时器。

1.4.2 用于上电清零(POC)的 Applilet 的配置

在使用的 Applilet 方案中，没有对话框来控制上电清零设定。Applilet 不在 option.inc 和 option.asm 文件中为 POC81 (0081H)区域提供一个值。这些文件为改变上电清零来编辑。

1.4.3 用于低电压检测其(LVI)的 Applilet 的配置

选择“LVI”按键进入 LVI 配置对话框。



选择“Enable detection operation”选项来使 Applilet 生成 LVI 检测器代码。

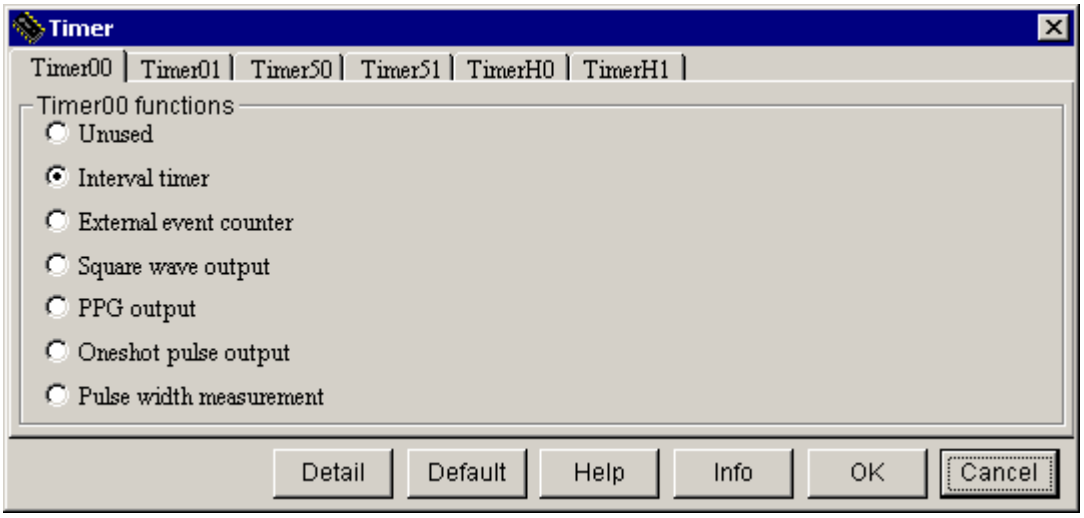
为了示范程序，我们将为示范改变缺省 LVI 设定。在“Voltage detection selection”中，我们选择“Power voltage (VDD) detect”来设定开始 LVI 检测器检测 VDD 电压。这控制 LVISEL 位的初始设定。

在“Detection operation mode selection”中，我们选择 “Generate interrupt signal”来让 LVI 产生 INTLVI 中断来替代复位。这将控制 LVIMD 位的初始设定。选择这些选项，可以选择 INTLVI 中断的优先级，我们使用缺省低优先级中断组。

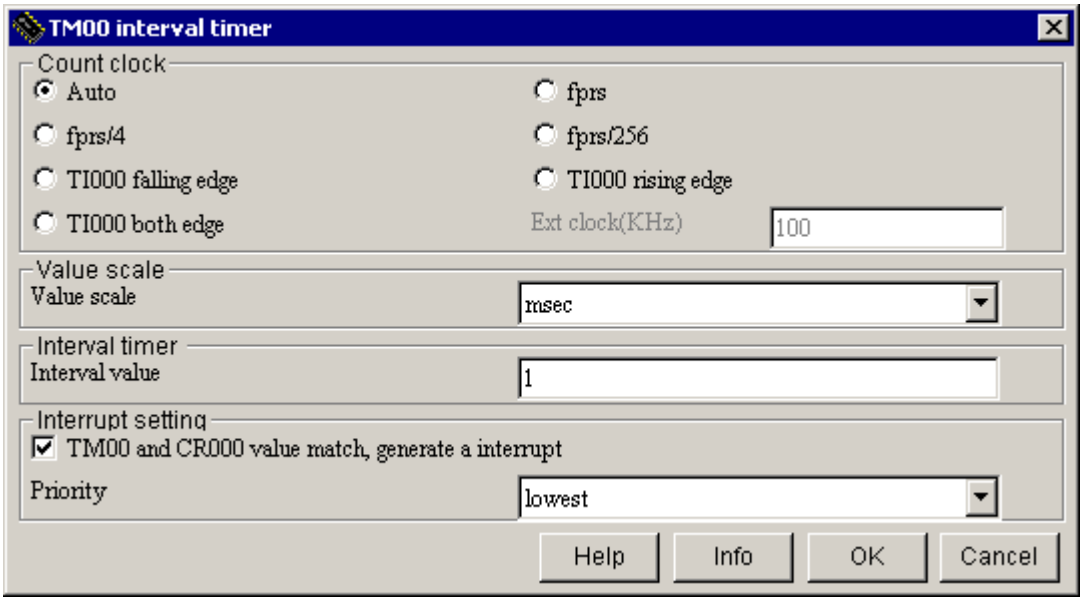
我们选择 VDD 电压电平之后，“Detection level selection”区域有效。我们选择 3.91V 作为初始选择。这将控制 LVIS 寄存器的初始写入值。

1.4.4 用于产生 1 毫秒间隔中断的定时器 TM00 的 Applilet 的配置

选择“Timer”按键进入一个展示变量定时器模块的对话框。“Timer00”选择定时器 00，并且定时器设为用于间隔定时器。



一旦选择使用定时器 00，在选择模式下用于定时器 00 设定的“Detail”按键就就有效。点击“Detail”按键进入详细设置定时器 00 间隔定时器的对话框。

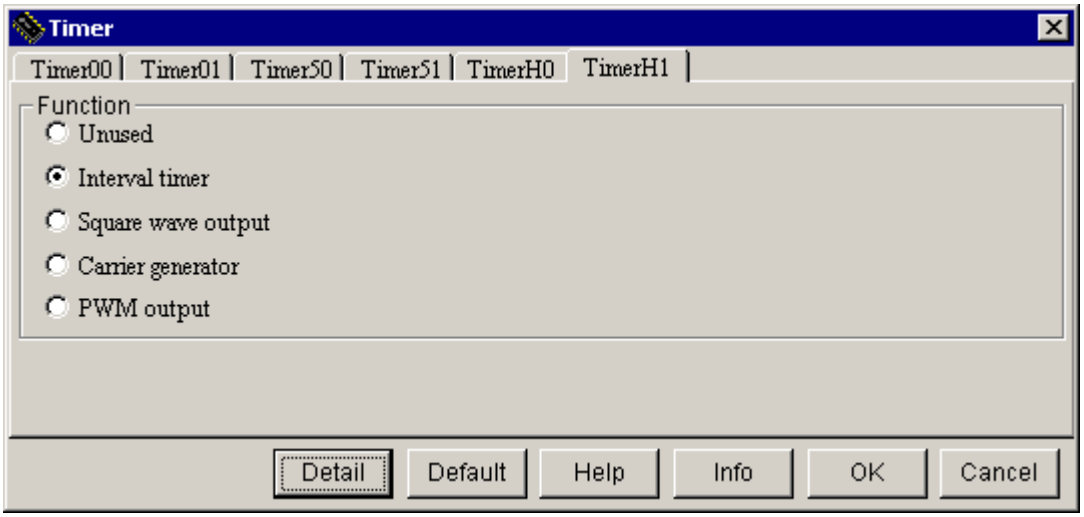


在这里选择定时器 00 的详细操作。我们需要定时器每 1 毫秒产生一个中断，所以设定 value scale 为“msec” (毫秒)，设定 interval value 为 1。count clock 设定左边的 “Auto”；这意味着 Applilet 选择适当的定时器始终选择寄存器的设定来产生 1 毫秒间隔。

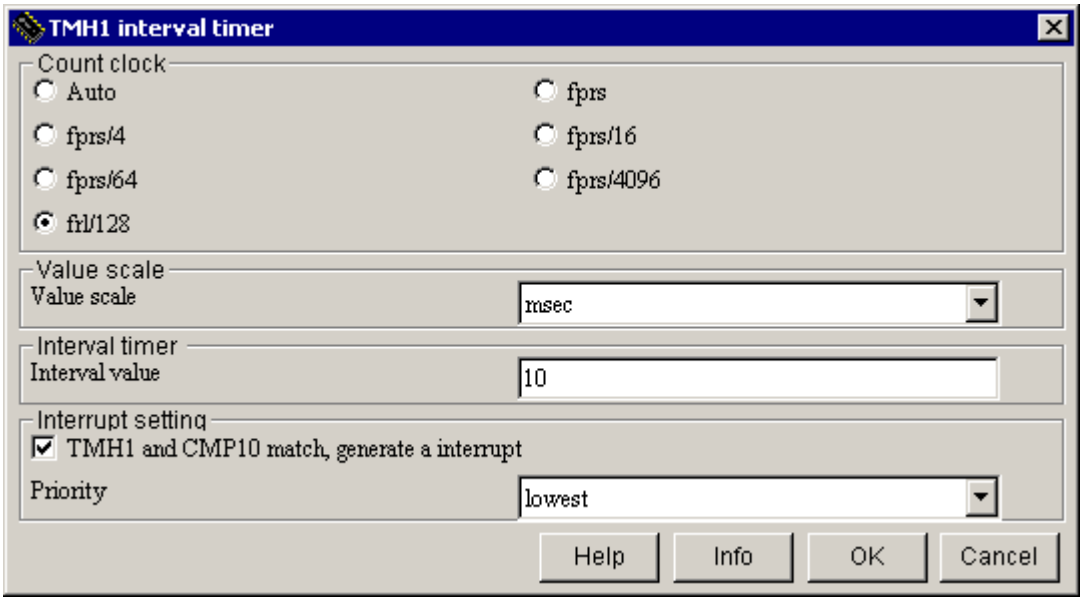
一旦我们需要的生成定时器中断设定好，我们检测检测框在 TM00(定时器计数寄存器)和 CR000(定时器比较寄存器)相等时生成的中断。这个中断的优先级设定为最低，来保持低优先级中断组。

1.4.5 用于产生 10 毫秒间隔中断的定时器 TMH1 的 Applilet 的配置

在 Timer 窗口，TimerH1 选项卡选择 TMH1 设置为间隔定时期。



一旦选择使用 TMH1，在选择模式下用于 TMH1 设定的“Detail”按键就就有效。点击“Detail”按键进入详细设置 TMH1 间隔定时器的对话框。



TMH1 详细操作的选择如下我们需要定时器运行在和看门狗定时器相同的时钟下，低速振荡器，所以我们选择 frl/128 作为计数时钟。我们需要定时器每 10 毫秒产生一个中断，所以设定 value scale 为“msec” (毫秒)，设定 interval value 为 10。Applilet 将计算出用于 CMP01 定时器比较寄存器的设定，来产生一个 10 毫秒间隔。(实际的选择值是 9.6 毫秒间隔，是最接近 10 的一个值。)

一旦我们需要的生成定时器中断设定好，我们检测检测框在 TMH1 (定时器计数寄存器)和 CR01(定时器比较寄存器)相等时生成的中断。注意对话框中错误的显示寄存器为“CMP10”。

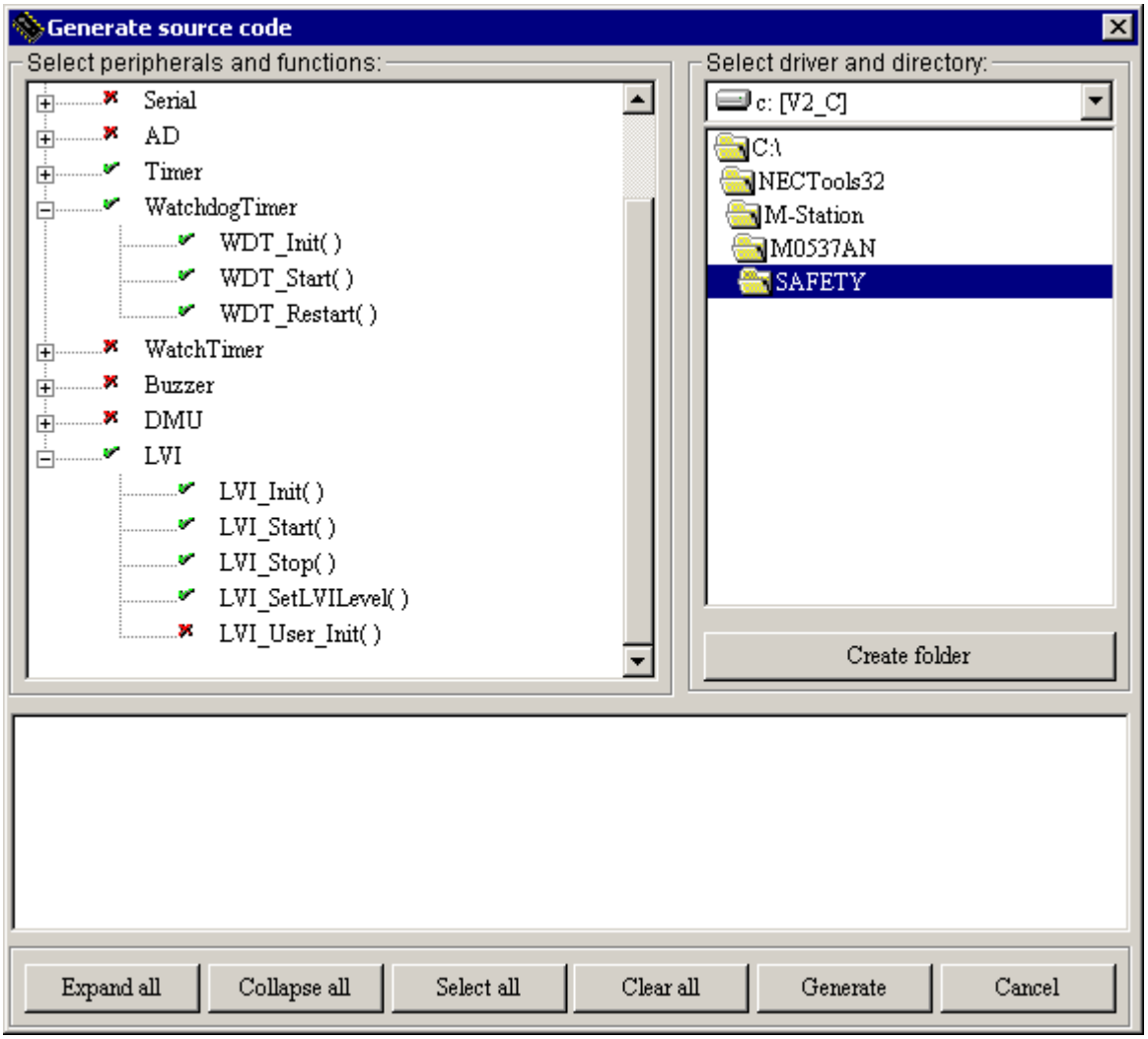
1.4.6 Applilet 生成代码，选择选项功能

一旦设定变量对话框，选择“Generate code”选项。Applilet 将显示生成的外围和功能，并且允许你去选择一个目录来保存源代码。

当 Applilet 初始化一个新工程，仅有选择部分用于生成的特殊外围有效。从对话框左边的功能树选择的外围展示什么功能有效，并且是否选择他们产生。如果选择了，这里展示一个检测标志，或者没有选择一个“x”。你可以通过点击这个标志来改变这个选项

为了看门狗定时器，仅选择 WDT_Init()。在我们的示范程序中，我们同样选择 WDT_Start()和 WDT_Restart()。

为了 LVI，仅选择 LVI_Init()。在我们的示范程序中，我们同样选择 LVI_Start()，LVI_Stop()，和 LVI_SetLVILevel()。



当按下“Generate”按键，Applilet 在几种 C 语言源文件(extension .c)和头文件(extension .h)中创建代码，并且在对话框中显示文件列表。

要支持看门狗定时器，Applilet 生成 watchdogtimer.h，和 watchdogtimer.c。它同样生成汇编语言文件 option.inc 和 option.asm，来定义 flash 存储器区域中的选项字节。

要支持低电压检测，Applilet 生成 lvi.h，lvi.c 和 lvi_user.c。

要支持 TM00 和 TMH1，Applilet 生成 timer.h，timer.c 和 timer_user.c.

要支持用于按键输入和 LED 输出的 I/O 端口， Applilet 生成 port.h 和 port.c。

还产生一些其它的文件，包括一个空的主函数 main.c 文件。

1.4.7 用于看门狗定时器(WDT)和 POC 的 Applilet 生成文件和函数

为支持看门狗定时器生成的代码文件是 watchdogtimer.h 和 watchdogtimer.c。代码在文件 option.inc 和 option.asm 中控制选项字节和 POC 字节。这些内容如下。

1.4.7.1 Watchdogtimer.h

头文件 watchdogtimer.h 包括控制看门狗定时器的函数的声明。

1.4.7.2 Watchdogtimer.c

源文件 watchdogtimer.c 包括为看门狗定时器通过 Applilet 产生的以下函数：

void WDT_Init(void);
如果有必要，通过 SystemInit()来为看门狗初始化调用 WDT_Init()程序。一旦看门狗定时器设为通过 0080H 的选项字节控制，这个函数没有任何返回。

void WDT_Start(void);
WDT_Start()函数写入值 0xAC 到 WDTE 寄存器，来清除看门狗定时器溢出计数器。一旦看门狗定时器通过选项字节启动，这和 WDT_Restart()的代码一样。处理器控制看门狗定时器开始的寄存器，程序事实上是从看门狗定时器开始。

void WDT_Restart(void);
WDT_Restart()程序写入值 0xAC 到 WDTE 寄存器，来清除看门狗定时器溢出计数器。通过调用这个程序来阻止通过看门狗定时器产生复位。

1.4.7.3 Option.inc

源文件 option.inc 是一个汇编语言包含文件，包含在 option.asm 中。它包括了定义放在 0080H 选项字节，POC81 的 POC81 字节，和其他位置的值的 EQU 陈述。

为了示范程序，编辑 option.inc 文件来生成两种不同的文件：生成 option.inc，为 1.59V 模式设定 POC 字节为 0x00；option_POC27.inc，为 2.7V/1.59V POC 模式设定 POC81 字节为 0x01。

1.4.7.4 Option.asm

汇编语言源文件 option.asm 包括分配存储器地址和赋值的 DB(定义字节)陈述，以及段定义指向指定存储器位置。指定选项字节 0080H，POC81 字节 0081H，和分配其他，并且在 option.inc 中赋值定义。

为了示范程序，编辑 option.asm 文件来生成两种不同的文件：生成 option.asm as generated，包含在 option.inc；option_POC27.asm，包含在修正 option_POC27.inc。

1.4.8 用于低电压检测器(WDT)的 Applilet 生成文件和函数

用于 LIV 的生成代码支持文件 lvi.h，lvi.c，和 lvi_user.c。这些包括以下项目。

1.4.8.1 Lvi.h

头文件 lvi.h 包括用于函数初始化，开始，停止，和配置以及停止 LVI 检测器的声明。

用于 LVI_Start_R()程序的声明附加在 Applilet 生成代码中。

1.4.8.2 Lvi.c

源文件 lvi.c 包括 Applilet 生成的用于 LVI 检测器操作的下列函数：

void LVI_Init(void);
LVI_Init()程序在 AppliletLVI 对话框中按规定初始化 LVI 检测器。

void LVI_Start(void);
LVI_Start()程序开始用于中断模式的 LVI 检测器操作。

void LVI_Stop(void);
LVI_Stop()程序通过清除 LVION 位停止 LVI 检测器以及通过设定 LVIMK 位来禁止 INTLVI 中断。

MD_STATUS LVI_SetLVILevel(enum LVILevel level);
LVI_SetLVILevel(level)程序写入电平参数到 LVIS 寄存器，来设置 VDD 检测电压。Lvi.h 定义用于电平的符号值； LVILevel1 是 1，LVILevel2 是 2，等等。

1.4.8.3 Lvi_user.c

源文件 lvi_user.c 包括用于用户代码的抽头函数。这些函数在生成代码时为空，来允许用户增加指定应用代码。

__interrupt void MD_INTLVI(void);
这是一个用于 LVI 检测器中断 INTLVI 的中断服务程序，当 LVI 单位在中断模式下时，当电压检测下降到检测阀值之下时产生。这将检测 VDD 电压或者 EXLVI 引脚电压，这由 LVISEL 位的状态来决定。

通过 Applilet 生成，中断服务程序是空。为了示范程序，代码增加通过调用 LVI_Stop()来停止 LVI 检测器，以及在 LED 上显示“LI”来指示一个 LVI 中断。

void LVI_Start_R(void);
LVI_Start_R()程序不通过 Applilet 生成；在示范中为了使用它如何在复位模式中开始 LVI 单位，从 LVI_Start()被修改。请参见下一节。

1.4.8.4 在中断和复位模式中用于 LVI 操作的 Applilet 代码

用于在指定中断模式中 LVI 检测器检测操作是否正确的代码通过 Applilet 产生。LVI_Init()中的操作频率是：

LVION = 0;	禁止 LVI 单位
LVISEL=x;	选择 VDD 或者 EXLVI 模式
LVIMD=0;	选择中断模式
---	设定中断优先级，如果使用的话设定用于 VDD 电平的 LVIS
LVION=1;	开启 LVI 单位
(delay)	最小等待 10 毫秒
LVIMK=0;	解除屏蔽 LVI 中断

LVI_Start()包括相似代码；它屏蔽中断和清楚标志，设定 LVION，延迟，和解除屏蔽中断。

然而，为了使用的 Applilet 方案，如果为 LVI 选择操作的复位模式，LVI_Init()和 LVI_Start()代码将包括一个镜像错误。

两个程序中，在为复位模式设定 LVIMD 位为 1 之后设定 LVION 位。LVI 检测器电压稳定后在不适当的电平下这将导致一个立即复位。复位模式的正确操作，LVIMD 位不应当设置来选择复位模式，直到设置 LVION 位并且完成延迟来稳定检测器电压。LVI_Init()中操作的适当频率应当是：

LVION = 0;	禁止 LVI 单位
LVISEL=x;	选择 VDD 或者 EXLVI 模式

LVIMD=0;	临时选择中断模式
---	设定中断优先级，如果使用的话设定用于 VDD 电平的 LVIS
LVION=1;	开启 LVI 单位
(delay)	最小等待 10 毫秒
LVIMD=1;	设定复位模式

如果在复位模式中为 LVI 控制使用 Applilet，请检测在 LVI_Init()和 LVI_Start()程序中的适当的频率。

1.4.9 用于 TM00 和 TMH1 的 Applilet 生成文件和函数

用于 TM00 和 TMH1 的生成代码支持文件 timer.h, timer.c 和 timer_user.c。这些包括以下项目。

1.4.9.1 Timer.h

头文件 timer.h 包括用于函数控制的定时器，以及的用于定时器值的定义的声明。头文件 macrodriver.h，用于所有的 Applilet 生成代码，同样定义一些数据类型和值，例如通过医学函数返回的 MD_STATUS 值。

要使全局变量 **g_tmh1_count** 在其它程序中有效，增加这个变量的外部声明到 Applilet 生成文件中去。

1.4.9.2 Timer.c

源文件 Timer.c 包括 Applilet 生成的用于 TM00 和 TMH1 的下列函数：

void TM00_Init(void);
TM00_Init()程序在 Applilet 定时器 00 详细对话框中按指定初始化定时器 00 外围。

void TM00_Start(void);
TM00_Start()程序通过允许寄存器和允许中断 INTTM000 来开始定时器 00 操作。

void TM00_Stop(void);
TM00_Stop()程序通过禁止定时器和禁止定时器中断来停止定时器 00 操作。这个程序在示范程序中未使用。

MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg, USHORT array_num);
TM00_ChangeTimerCondition()函数在定时器 00 比较寄存器，CR000 和 CR010，中改变值，因此改变用于定时器 00 的间隔。

array_reg 参数是一个用于要放在一个或者其它比较寄存器中的值的阵列的指针；array_num 参数是 1 (用于选择 CR000)或 2 (用于选择 CR010 和 CR000)。因为我们不需要改变定时器 00 的间隔时间，这个程序在示范程序中未使用。

void TMH1_Init(void);
TMH1_Init()程序在 Applilet TMH1 详细对话框中按指定初始化 TMH1 外围。

void TMH1_Start(void);
TMH1_Start()程序通过允许定时器和允许中断 INTTMH1 来开始 TMH1 操作。

void TMH1_Stop(void);
TMH1_Stop()程序通过禁止定时器和禁止中断来停止 TMH1 操作。这个程序在示范程序中未使用。

MD_STATUS TMH1_ChangeTimerCondition(UCHAR* array_reg, UCHAR array_num);

TMH1_ChangeTimerCondition()函数在定时器 H1 比较寄存器，CMP01 和 CMP11，中改变值，因此改变用于定时器 H1 的间隔。

array_reg 参数是一个用于要放在一个或者其它比较寄存器中的值的阵列的指针；array_num 参数是 1 (用于选择 CMP01)或 2 (用于选择 CMP01 和 CMP11).

1.4.9.3 Timer_user.c

源文件 timer_user.c 包括用于用户代码的抽头函数。这些函数在生成代码时为空，来允许用户增加指定应用代码。

__interrupt void MD_INTTM000(void);
这是一个用于定时器 00 中断 INTTM000 的中断服务程序，当 TM00 和 CR000 值相等时产生。一旦定时器开始，这个中断将每 1 毫秒产生一次。

通过 Applilet 生成，中断服务程序是空。要使用间隔定时器同过检每毫秒测它们的值来去抖动按键，代码增加到 MD_INTTM000()来调用 sw_isr()函数来进行按键检测。

__interrupt void MD_INTTMH1(void);
这是一个用于定时器 H1 中断 INTTMH1 的中断服务程序，当 TMH1 和 CMP01 值相等时 ichansheng 。当定时器开始设定，这个中断将每 9.6 毫秒产生一次。

通过 Applilet 生成，中断服务程序是空。要使用间隔定时器来复位看门狗定时器，增加代码来调用 WDT_Restart()。要提供一个用于看门狗定时器示例子程序来指示 INTTMH1 产生的速度的方法，增加代码来加一全局变量 **g_tmh1_count**。

1.4.10 用于端口初始化的 Applilet 生成文件和函数

为支持 I/O 端口生成代码，在“Digital I/O Port”对话框设定，文件 port.h 和 port.c。这些包括下列项目。

1.4.10.1 Port.h

头文件 port.h 包括了用于初始化端口的 PORT_Init()函数，和用于初始化端口出数锁存，端口模式寄存器，和上拉电阻的定义值的声明。例如，要定义端口 P0，值要写入到 P0，PM0，和 PU0。这些值在 port.h 中定义。

要改变用于端口的初始化值，在 port.h 中的定义可以编辑。为了示范程序，当在键扫程序中不使用作为输出时编辑 PORT_PU0 的值来设置 P04 和 P05 的上拉电阻。

1.4.10.2 Port.c

源文件 port.c 包括了下面通过 Applilet 生成的用于端口初始化的函数：

void PORT_Init(void);
PORT_Init()程序通过设定端口输出锁存，端口模式寄存器，和每个端口的上拉电阻来初始化全部设备 I/O 端口，这些值在 port.h 中定义。

1.4.11 其它 Applilet 生成文件

为了示范程序，Applilet 生成其他几种源文件。这些文件和他们的功能如下所示。

文件	功能
Macrodriver.h	用于 Applilet 生成程序的通用头文件
Systeminit.c	用于 SystemInit()和 hdwinit()函数
Main.c	主程序函数
System.h	时钟相关定义
System.c	Clock_Init()函数

1.4.12 不由 Applilet 生成的示范程序文件

示范程序有文件是由 Applilet 生成的。

示范程序也有文件不是由 Applilet 生成的，列表如下。

文件	功能
Sw_0537.h	用于按键输入的头文件
Sw_0537.c	按键输入的代码和去抖动
Led_0537.h	用于 7 段 LED 结构和函数的头文件
Led_0537.c	在 7 段 LED 中的显示数据代码

1.5 示范平台

当本文档准备好时从 NEC 开发工具中选择的示范平台可用。通常用户可能需要通过使用标准通用元件来复制和 NEC 微控制器性能相同的硬件。

1.5.1 源

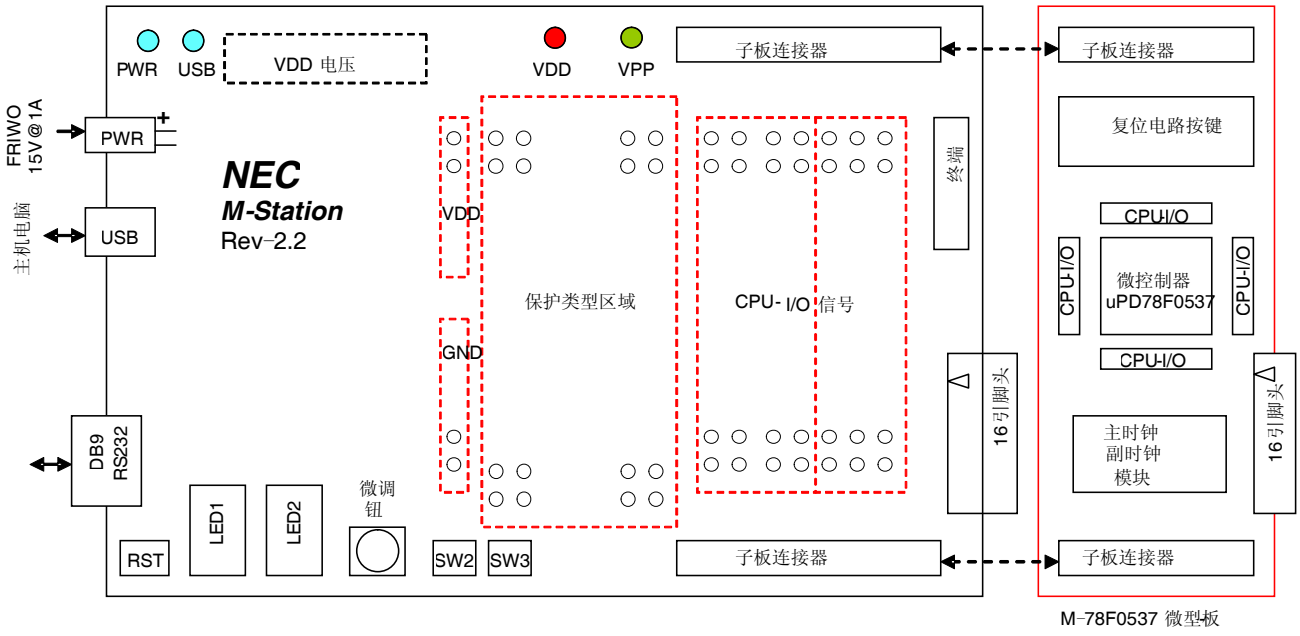
要示范程序，使用下列源：

- M-78F0537 微型板，板载 uPD78F0537 8 位微控制器
- M-Station II 评价系统，使用 M-Station II 源：
  - RST 切换生成低压复位
  - 7 段 LED 显示 LED1 和 LED2
  - 按键按键 SW2 和 SW3
  - 用于 VDD-GND 连接的电位器

在 VDD_FLASH (JP1.1)和 GND 之间增加 100 欧姆电位器；电位器的抽头连接到 VDD (JP1.2)。这允许一个可变的 VDD 电压。

VDD 和 GND 之间的标准 M-Station-II 电位器，通过缺省 uPD78F0537 模拟输入 P20/ANI0 连接，同样通过增加 J1_B.17(P120/EXLVI)和 MH_009(电位器抽头)和之间的连线连接到。

上面详细的硬件列表，请参考适当的用户手册。



1.5.2 程序示范

假定硬件正确的配置并且 uPD78F0537 微控制器编程了示范程序代码，示范如下。

1.5.2.1 外部复位示范

- 应用 VDD 电压在 Vpoc 阀值之上，在 LED 上显示“00”
- 按下 SW2 或 SW3，可见“=1” (选择测试)
- 按下 RST 按键来声明复位；释放时，可见 LED 显示“00”

1.5.2.2 上电清零(POC)示范

有两个方案的示范程序。一个使用 option.inc 和 option.asm 文件，设定 POC 模式为 1.59 模式，Vpoc 的上电电压是 1.59V。另外一个使用 option_POC27.inc 和 option_POC27.asm 文件，设定 POC 模式为 2.7V/1.59V 模式，Vpoc 的上电电压是 2.7V，并且然后是 1.59V。

- 应用 VDD 电压低于 Vpoc 阀值；LED 显示为空
- 上升 VDD 电压高于 Vpoc 阀值；LED 显示为“00”
- 按下 SW2 或 SW3，LED 显示“=1”
- VDD 电压低于 1.59V；LED 显示为空(表示复位)
- 上升 VDD 电压高于 1.59V，LED 显示为“00”

1.5.2.3 低电压检测(LVI)示范

SW2 或 SW3 按下后，并且显示“=1”，按下 SW2 将改变显示为“=2”，“=3”，直到“=7”然后返回“=1”。这选择 7 种测试中的一种；按下 SW3 将执行测试。当按下 SW3，左边数字显示测试数字，右边数字显示空。测试 1～6 是 LVI 测试，测试 7 是看门狗定时器示范：

测试	LED	测试功能	功能操作
1	“1 “	LVI_Reset_239()	VDD < 2.39V 时设定 LVI 到 RESET
2	“2 “	LVI_Reset_393()	VDD < 3.93V 时设定 LVI 到 RESET
3	“3 “	LVI_Int_239()	VDD < 2.39V 时设定 LVI 到中断
4	“4 “	LVI_int_393()	VDD < 3.93V 时设定 LVI 到中断
5	“5 “	EXLVI_Reset()	EXLVI < 1.21V 时设定 LVI 到 RESET
6	“6 “	EXLVI_Int()	EXLVI < 1.21V 时设定 LVI 到中断
7	“7 “	WDT_Demo()	WDT 计数器溢出时显示 RESET

VDD 低压模式时 LVI 复位示范

- 应用 VDD 电压高于 3.93V
- 选择测试“=1”或“=2”，按下 SW3；显示“1”或“2”
- 减少 VDD 电压低于设定电压
- 在适当电压下 LED 显示空
- 上升 VDD 电压高于设定电压
- LED 显示“01”指示 LVI 导致的复位

VDD 低压模式时 LVI 中断示范

- 应用 VDD 电压高于 3.93V
- 选择测试“=3”或“=4”，按下 SW3；显示“3”或“4”
- 减少 VDD 电压低于设定电压
- 在适当电压下 LED 显示“LI”
- 按 SW2 来返回测试菜单

EXLVI 低压模式时 LVI 复位示范

- 应用 VDD 电压
- 设定用于 EXLVI 的电位器高于 1.21V
- 选择测试“=5”；按下 SW3；显示“5”
- 降低 EXLVI 电压低于 1.21V
- 在适当电压下 LED 显示空
- 上升 EXLVI 电压高于 1.21V
- LED 显示“01”指示 LVI 导致的复位

EXLVI 低压模式时 LVI 中断示范

- 应用 VDD 电压
- 选择测试“=6”；按下 SW3；显示“6”
- 降低 EXLVI 电压低于 1.21V
- 在适当电压下 LED 显示“LI”
- 上升 EXLVI 电压高于 1.21V
- 按 SW2 来返回测试菜单

1.5.2.4 看门狗定时器示范

初始化时，设定看门狗定时器溢出时间为 68 毫秒。定时器 H1 初始化为每 9.6 毫秒产生一次中断，并且中断服务程序中清除看门狗计数器。

- 选择“=7”，按下 SW3
- 显示“71”，右边数字快速闪烁

此时，定时器 H1 中断 INTTMH1 每 9.5 毫秒产生一次，并且清楚看门狗定时器。

- 再次按下 SW3
- 显示“72”，“73”，等等，右边数字闪烁的越来越慢

每按一下 SW3，定时器 H1 的间隔增加 9.6 毫秒。闪烁是通过在用于 INTTMH1 的中断服务程序中加一一个变量产生。间隔增加，INTTMH1 中断的频率越低，所以闪烁的越慢，

下表是按 SW3 次数，设定定时器 H1 价格，和 LED 显示。

SW3 按键	定时器 H1 间隔	LED 显示
1	9.6 毫秒	“71”，1 快速闪烁
2	19.2 毫秒	“72”，2 闪烁的更慢
3	28.8 毫秒	“73”，3 闪烁的更慢
4	38.4 毫秒	“74”，4 闪烁的更慢
5	48.0 毫秒	“75”，5 闪烁的更慢
6	57.6 毫秒	“76”，6 闪烁的更慢
7	67.2 毫秒	“77”，7 闪烁的更慢
8	76.8 毫秒	“10”；看门狗定时器产生的复位

当定时器 H1 的间隔时间超过 68 毫秒，看门狗定时器没有及时的清除，并且计数溢出，产生一个看门狗定时器复位。程序再次开始，为 RESF 显示“10”，展示了设置的 WTDRF 标志，显示看门狗定时器导致了复位。

此时，程序再次初始化，所以定时器 H1 间隔再次设为 9.6 毫秒，所以会在溢出前清除看门狗定时器。

1.5.3 使用 M-Station VDD 选择替代电位器的示范

M-Station-II 有可选的 VDD 电平，可能有限制的用于上面的示范方案。VDD 设定按键允许 VDD 来关或者开；当关时，可以从设定电压 2.0V, 2.5V, 3.0V, 3.3V, 4.0V, 4.5V, 和 5.0V 中选择 VDD 电压。

上电清零示范：

- 示范程序方案中具有 POC 模式 2.7/1.59V 的程序
- 通过按下 VDD 设定按键 5 秒来关闭 VDD
- 设定 VDD 电压选择为 2.5V
- 通过按下 VDD 设定按键 5 秒来打开 VDD
- 观察 LED 显示闪烁(VDD 低于 Vpoc 阀值)
- 关闭 VDD
- 设定 VDD 电压为 3.0V
- 打开 VDD
- 观察 LED 显示为“00” (VDD 高于 Vpoc 阀值)

以具有 POC 模式 1.59V 示范程序方案重编程设备；观察在 VDD 从 2.0V～5.0V 变化的全部的值的程序运行。

注: 当改变 POC81 字节 0081H 的值，必须全部擦除 uPD78F0537(片模式擦除)。通过使用 block 模式编程，或者使用自编程模式部分擦除，将不会改变 POC81 字节的值。

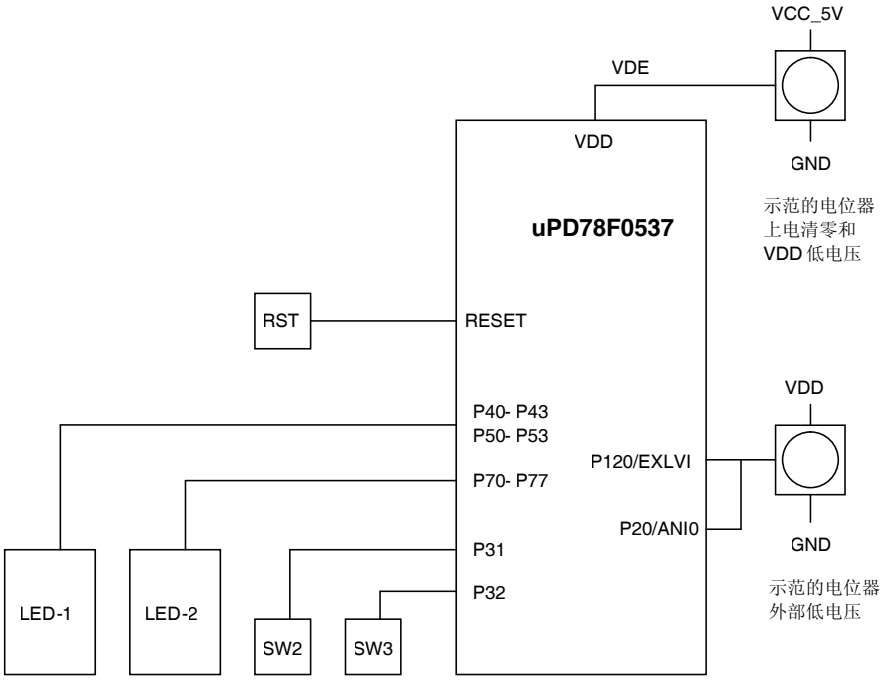
要示范低电压检测

- 在 5.0V 打开 VDD
- 运行测试 =1, =2, =3, =4
- 观察没有 LVI 产生(显示没有为空或者“LI”)

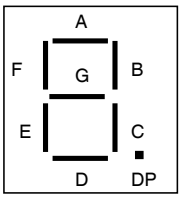
- 在 3.3V 关闭 VDD
- 运行测试 =1, =2, =3, =4
- 在 tests =1 或 =3 时观察没有 LVI (VDD 高于 2.39V)
- 在 test =2 时观察 LED 显示空 (VDD 低于 3.93V，复位)
- 在 test =4 时观察 LED 显示“LI” (VDD 低于 3.93V，中断)

- 确保变成具有 POC 示范程序模式 1.59V 的示范程序
- 关闭 VDD，然后在 2.0V 打开(显示将暗淡)
- 运行测试 =1, =2, =3, =4
- 在 tests =1 和 =3 时观察 LED 显示闪烁 (VDD 低于 2.39V，复位)
- 在 tests =3 和 =4 时观察 LED 显示“LI” (VDD 低于 2.39V，中断)

1.6 硬件框图



LED 显示连接如下所示。



LED-1 和 LED-2

段	LED-1	LED-2
A	P40	P70
B	P41	P71
C	P42	P72
D	P43	P73
E	P50	P74
F	P51	P75
G	P52	P76
DP	P53	P77

1.7 软件模块

下列文件整理了用于示范程序的软件模块。下列表中的文件是由 Applilet 生成，用于编辑和创建示范程序。

表中的这些文件位于附录中。

文件	目的	由 Applilet 生成	由用户修改
Main.c	主程序	是	是
Macrodriver.h	通过 Applilet 常规定义使用	是	否
System.h	时钟相关定义	是	否
Systeminit.c	SystemInit()和 hdwinit()函数	是	否
System.c	Clock_Init()函数	是	否
Lvi.h	LVI 相关定义	是	是[注1]
Lvi.c	LVI 函数	是	是[注1]
Lvi_user.c	用于 LVI 中断处理的用户代码	是	是[注1]
Port.h	端口相关定义	是	否
Port.c	Port_Init()函数	是	否
Timer.h	定时器相关定义	是	是[注2]
Timer.c	定时器函数	是	否
Timer_user.c	用于定时器中断处理的用户代码	是	是[注2]
Watchdogtimer.h	WDT 相关定义	是	否
Watchdogtimer.c	WDT 函数	是	否
Option.inc	选项字节，POC，和安全定义	是	否
Option.asm	选项字节，POC，和安全数据	是	否
Led_0537.h	LED 显示定义	--	是
Led_0537.c	LED 显示函数	--	是
Sw_0537.h	按键输入定义	--	是
Sw_0537.c	按键输入函数	--	是

注 1: 编辑 Lvi.h 来增加 LVI_Start_R()程序的声明。编辑 Lvi.c 来注释在 LVI_Init()程序结束允许 LVI 中断。编辑 Lvi_user.c 来在 MD_INTLVI()程序中增加处理 INTLVI 中断的代码，并且增加 LVI_Start_R()程序。

注 2: 编辑 Timer.h 来增加全局变量 **g_tmh1_count** 的声明。编辑 Timer_user.c 来增加在用于按键去抖动的 MD_INTTM000()程序中处理 INTTM00 中断的代码，增加在用于清除看门狗定时器的 MD_INTTMH1()程序中处理 INTTMH1 中断的代码，并且增加全局变量 g_tmh1_count，用于跟踪 INTTMH1 中断的速度。

提供一个程序的案例，来设定上电清零函数为 2.7V 电平，提供下列文件：

文件	目的	由 Applilet 生成	由用户修改
Option_POC27.inc	选项字节，POC，和安全定义	是*	是
Option_POC27.asm	选项字节，POC，和安全数据	是*	是

从 option.inc 中编辑 Option_POC27.inc 来改变 POC81 值从 00H 到 01H，设定 POC 模式为 2.7V/1.59V 模式。从 option.asm 中编辑 Option_POC27.asm 来包含 option_POC27.inc 文件来替代 option.inc。除了这些改变，文件由 Applilet 生成。

2 附录A – 开发工具

本文档开发中使用下列软件和硬件工具。

2.1 软件工具

工具	版本	解释
用于 78K0KX2 的 Applilet	V1.51	用于 78K0/KE2 设备的源代码生成工具
PM Plus	V5.20	用于程序编译和链接的项目管理器
CC78K0	V3.60	用于 NEC 78K0 设备的 C 编译器
RA78K0	V3.70	用于 NEC 78K0 设备的汇编器
DF053764.78K	V2.00	用于 uPD78F0537_64 设备的设备文件

2.2 硬件工具

工具	版本	解释
M-Station 2	V2.1E	用于 NEC 微型示范板的基本平台
M-78F0537	V1.0	用于 uPD78F0537 的 NEC 微型板；CUP 芯片是是 uPD78F0537DGB

编辑 M-Station 2 来增加 JP1.1 (VDD_FLASH)和 GND 之间的电位器，它的抽头连接到 JP1.2 (VDE)来允许 uPD78F0537 微控制器的 VDD 电压可变。

一个 J1_B.17 (M-Station 信号 J1_043, M-78F0537 信号 P120/EXLVI)和 MH_009 (M-Station VDD 和 GND 之间的电位器)之间的连接，允许输入 EXLVI 输入的电压可变。

3 附录 B – 软件列表

注: 包含 option_POC27.inc 文件和 option_POC27.asm 来建立具有上电清零模式为 2.7V/1.59V 模式的示范程序的备用方案。

3.1 Main.c

```
/*
************************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 – 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名 : main.c
**  摘要 : 该文件执行主函数
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备 :    uPD78F0537
**
**  编译器 :  NEC/CC78K0
**
************************************************************************************
*/
/*
************************************************************************************
**  包含文件
************************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
#include "port.h"
#include "timer.h"
#include "watchdogtimer.h"
#include "lvi.h"

/* 增加非 Applilet 函数 */
#include "led_0537.h"
#include "sw_0537.h"

/*
************************************************************************************
**  宏定义
************************************************************************************
*/
#define      TEST_MAX     7

/* test function definitions */
void LVI_Reset_239(void);         /* 设定用于 2.39V 复位的 LVI */
void LVI_Reset_393(void);         /* 设定用于 3.93V 复位的 LVI */
void LVI_Int_239(void);           /* 设定用于 2.39V 中断的 LVI */
void LVI_Int_393(void);           /* 设定用于 3.93V 中断的 LVI */
void EXLVI_Reset(void);           /* 设定用于 1.51V 的 EXLVI 复位 */
void EXLVI_Int(void);             /* 设定用于 1.51V 的 EXLVI 中断 */
void WDT_Demo(void);       /* 看门狗定时器演示 */

/* 定义 PF_VF_RV 为函数指针，空参数，空返回 */
typedef void (*PF_VF_RV)(void);
```

```
PF_VF_RV menu[TEST_MAX] = {
       LVI_Reset_239,
       LVI_Reset_393,
       LVI_Int_239,
       LVI_Int_393,
       EXLVI_Reset,
       EXLVI_Int,
       WDT_Demo };

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     主函数
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void main( void )
{
unsigned char resf_val;    /* 从 RESF 读取值 */
unsigned char test;        /* 选择测试运行 */
unsigned char sw_val;      /* 按键值 */

       IMS = MEMORY_IMS_SET;
       IXS = MEMORY_IXS_SET;
       /* TODO. 增加到用户代码 */

       TMH1_Start();
       WDT_Start();

       led_init();         /* 初始化 LEDs */
       sw_init();          /* 初始化按键变量 */
       sw_set_debounce(10);/* 为 10 毫秒稳定时间设定去抖动计数器为 10 */

       TM00_Start(); /* 为按键去抖动开始定时器 */

       resf_val = RESF;    /* 读取复位标志寄存器 */
       led_dig(resf_val);  /* 显示变量 */
       while (SW_LU_RU == sw_get())
              ;                           /* 等待按键按下 */
       while (SW_LU_RU != sw_get())
              ;                           /* 等待按键抬起 */

       test = 1;                  /* 设定初始测试号码 */
       while(1){
              led_out_left(LED_PAT_EQUAL);      /* 显示等号 */
              led_dig_right(test);              /* 显示要执行的测试号码 */

              while (SW_LU_RU == sw_get())
                     ;                                                  /* 等待按键按下 */
              sw_val = sw_get();                        /* 获得按键值 */

              if (sw_val == SW_LD_RU) {
                     /* SW2 (left) is down, right (SW3) is up */
                     test = test + 1;                   /* 测试号码加一 */
                     if (test > TEST_MAX)
                            test = 1;                           /* 如果最大则返回 */
                     led_dig_right(test);       /* 立即更新显示 */
              }

              if (sw_val == SW_LU_RD) {
```

```
                    /* SW2 抬起, SW3 按下 */
                    while (SW_LU_RU != sw_get())
                          ;                                          /* 等待按键抬起 */
                    led_dig_left(test);               /* 左边显示测试 */
                    led_out_right(LED_PAT_BLANK);     /* 右边显示空 */

                    (menu[test-1])();                          /* 执行函数 */
              }

              while (SW_LU_RU != sw_get())
                    ;                                          /* 等待按键抬起 */
       } /* while (1)循环结束 */
} /* main()结束 */

void LVI_Reset_239(void)
{
       /* 为 2.39V 复位设定 LVI */
       LVI_Stop();          /* 确保已经停止 */
       LVISEL = 0;          /* 通过 LVIS 寄存器选择 LVI 检测 VDD 电平作为设定 */
       LVIMD = 0;           /* 设定 LVI 模式来临时中断 */
       LVI_SetLVILevel(LVI_Level12);     /* 为电平 12(2.39V)设定 */
       LVI_Start_R();       /* 开始不具有允许中断的 LVI 检测 */
       LVIMD = 1;           /* 设定 LVI 复位模式 */

       /* 返回前等待按键按下和释放 */
       while (SW_LU_RU == sw_get())      ;
       while (SW_LU_RU != sw_get())      ;
}


void LVI_Reset_393(void)
{
       /* 为 3.93V 复位设定 LVI */
       LVI_Stop();          /* 确保已经停止 */
       LVISEL = 0;          /* 通过 LVIS 寄存器选择 LVI 检测 VDD 电平作为设定 */
       LVIMD = 0;           /* 设定 LVI 模式来临时中断 */
       LVI_SetLVILevel(LVI_Level2);      /* 为电平 12(2.39V)设定 */
       LVI_Start_R();       /* 开始不具有允许中断的 LVI 检测 */
       LVIMD = 1;           /* 设定 LVI 复位模式 */

       /* 返回前等待按键按下和释放 */
       while (SW_LU_RU == sw_get())      ;
       while (SW_LU_RU != sw_get())      ;
}


void LVI_Int_239(void)
{
       /* 为 2.39V 中断设定 LVI */
       LVI_Stop();          /* 确保已经停止 */
       LVISEL = 0;          /* 通过 LVIS 寄存器选择 LVI 检测 VDD 电平作为设定 */
       LVIMD = 0;           /* 设定 LVI 模式来中断 */
       LVI_SetLVILevel(LVI_Level12);     /* 为电平 12(2.39V)设定 */
       LVI_Start();  /* 开始 LVI 检测并且允许中断 */

       /* 返回前等待按键按下和释放 */
       while (SW_LU_RU == sw_get())      ;
       while (SW_LU_RU != sw_get())      ;
}


void LVI_Int_393(void)
{
       /* 为 3.93V 中断设定 LVI */
       LVI_Stop();          /* 确保已经停止 */
       LVISEL = 0;          /* 通过 LVIS 寄存器选择 LVI 检测 VDD 电平作为设定 */
       LVIMD = 0;           /* 设定 LVI 模式来中断 */
       LVI_SetLVILevel(LVI_Level2);      /* 为电平 12(2.39V)设定 */
```

```
        LVI_Start();  /* 开始 LVI 检测并且允许中断 */

        /* 返回前等待按键按下和释放 */
        while (SW_LU_RU == sw_get())      ;
        while (SW_LU_RU != sw_get())      ;
}


void EXLVI_Reset(void)
{
        /* 为 1.51V EXLVI 复位设定 */
        LVI_Stop();          /* 确保已经停止 */
        LVISEL = 1;          /* 选择 LVI 检测 EXLVI 电平高于 1.21V */
        LVIMD = 0;           /* 设定 LVI 模式来临时中断 */
        LVI_Start_R();       /* 开始不具有允许中断的 LVI 检测 */
        LVIMD = 1;           /* 设定 LVI 复位模式 */

        /* 返回前等待按键按下和释放 */
        while (SW_LU_RU == sw_get())      ;
        while (SW_LU_RU != sw_get())      ;
}


void EXLVI_Int(void)
{
        /* 为 1.51V EXLVI 中断设定 */
        LVI_Stop();          /* 确保已经停止 */
        LVISEL = 1;          /* 选择 LVI 检测 EXLVI 电平高于 1.21V */
        LVIMD = 0;           /* 设定 LVI 模式来中断 */
        LVI_Start();  /* 开始 LVI 检测并且允许中断 */

        /* 返回前等待按键按下和释放 */
        while (SW_LU_RU == sw_get())      ;
        while (SW_LU_RU != sw_get())      ;
}


void WDT_Demo(void)
{
UCHAR count;
UCHAR reg[2];
UCHAR sw_val;
        /* 看门狗定时器演示 */
        count = 1;                                                  /* 定时器的数目加一 */
        reg[0] = TM_TMH1_INTERVALVALUE;          /* 为一次加一设定 TMH1 */
        TMH1_ChangeTimerCondition(&reg[0],1);

        while (1) {
                while (SW_LU_RU == (sw_val = sw_get())) {
                        /* 右边 LED 闪烁到按键按下 */
                        /* g_tmh1_count 技术将要增加的速度 */
                        /* 定时其计数增加，并且闪烁变慢 */
                        if (g_tmh1_count & 0x04)
                                led_out_right(LED_PAT_BLANK);
                        else
                                led_dig_right(count);
                }
                /* 按下按键 */
                while (SW_LU_RU != sw_get())
                        ;       /* 等待按键抬起 */

                if (sw_val == SW_LU_RD) {
                        /* 如果按下右边按键，加一计数，并且增加到间隔 */
                        count = count + 1;
                        reg[0] = reg[0] + (TM_TMH1_INTERVALVALUE + 1);
                        TMH1_ChangeTimerCondition(&reg[0],1);
                        /* 现在定时器间隔 = (计数) * (基本间隔) */
                }
```

```
            if (sw_val == SW_LD_RU) {
                  break; /* 退出循环 */
            }
      } /* while (1)循环结束 */
}
```

3.2 Macrodriver.h

```
/*
************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名 : macrodriver.h
**  摘要 : 这是通用头文件
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备 :    uPD78F0537
**
**  编译器:  NEC/CC78K0
**
************************************************************************
*/
#ifndef      _MDSTATUS_
#define _MDSTATUS_

#pragma sfr
#pragma di
#pragma ei
#pragma NOP
#pragma HALT
#pragma STOP

/* 数据类型定义 */
typedef      unsigned long ULONG;
typedef      unsigned int  UINT;
typedef      unsigned short      USHORT;
typedef      unsigned char UCHAR;
typedef      unsigned char BOOL;

#define      ON     1
#define      OFF    0

#define      TRUE   1
#define      FALSE  0

#define IDLE  0                  /* idle status */
#define READ  1                  /* read mode */
#define WRITE 2                  /* write mode */

#define SET   1
#define CLEAR 0

#define MD_STATUS          unsigned short
#define MD_STATUSBASE            0x0
```

```
/* 状态列表定义 */
#define MD_OK                    MD_STATUSBASE+0x0   /* 设定寄存器完成 */
#define MD_RESET                 MD_STATUSBASE+0x1   /* 复位输入 */
#define MD_SENDCOMPLETE                MD_STATUSBASE+0x2   /* 完成发送数据 */
#define MD_OVF                         MD_STATUSBASE+0x3   /* 定时器计数溢出 */

/* 错误列表定义 */
#define MD_ERRORBASE       0x80
#define MD_ERROR           MD_ERRORBASE+0x0    /* 错误 */
#define MD_RESOURCEERROR   MD_ERRORBASE+0x1    /* 无源有效 */
#define MD_PARITYERROR           MD_ERRORBASE+0x2    /* UARTn 奇偶错误 */
#define MD_OVERRUNERROR          MD_ERRORBASE+0x3    /* UARTn 超限错误 */
#define MD_FRAMEERROR            MD_ERRORBASE+0x4    /* UARTn 帧错误 */
#define MD_ARGERROR        MD_ERRORBASE+0x5    /* 错误参数输入错误 */
#define MD_TIMINGERROR           MD_ERRORBASE+0x6    /* 错误时序操作错误 */
#define MD_SETPROHIBITED   MD_ERRORBASE+0x7    /* 禁止设定 */
#define MD_DATAEXISTS            MD_ERRORBASE+0x8    /* 要发送的下一个数据存放在 TXBn 寄存器 */
#define MD_SPT                   MD_STATUSBASE+0x8   /*IIC 停止*/
#define MD_NACK                  MD_STATUSBASE+0x9   /*IIC 无响应*/
#define MD_SLAVE_SEND_END  MD_STATUSBASE+0x10  /*IIC 从发送结束 */
#define MD_SLAVE_RCV_END   MD_STATUSBASE+0x11   /*IIC 从接受结束 */
#define MD_MASTER_SEND_END MD_STATUSBASE+0x12   /*IIC 主发送结束 */
#define MD_MASTER_RCV_END  MD_STATUSBASE+0x13  /*IIC 主接收结束 */

/* 主时钟和副时钟作为时钟源 */
enum ClockMode { HiRingClock, SysClock };

/* 用于 IMS 和 IXS 的值 */
#define  MEMORY_IMS_SET          0xCC
#define  MEMORY_IXS_SET          0x00
/* 清楚 IO 寄存器位并且设定 IO 寄存器位 */
#define ClrIORBit(Reg, ClrBitMap)Reg &= ~ClrBitMap
#define SetIORBit(Reg, SetBitMap)Reg |= SetBitMap

enum INTLevel { Highest, Lowest };


#define      SYSTEMCLOCK  8000000
#define      SUBCLOCK     32768
#define      MAINCLOCK    8000000
#define      FRCLOCK      8000000
#define      FRCLOCKLOW   240000


#endif
```

3.3 System.h

```
/*
*********************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
```

```
**
**  文件名 :   system.h
**  摘要 :     该文件执行用于 SYSTEM 模块的设备驱动
**  APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备 :     uPD78F0537
**
**  编译器 :   NEC/CC78K0
**
*****************************************************************************
*/
#ifndef       _MDSYSTEM_
#define       _MDSYSTEM_
/*
*****************************************************************************
**  宏定义
*****************************************************************************
*/
#define       CG_X1STAB_SEL 0x5
#define       CG_X1STAB_STA 0x1f
#define       CG_CPU_CLOCKSEL     0x0

enum   CPUClock   {   SystemClock,   Sys_Half,   Sys_Quarter,   Sys_OneEighth,   Sys_OneSixteen,
Sys_SubClock };
enum PSLevel { PS_STOP, PS_HALT };
enum StabTime { ST_Level0, ST_Level1, ST_Level2, ST_Level3, ST_Level4 };

void  Clock_Init( void );

#endif
```

3.4 Systeminit.c

```
/*
*****************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2  8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名 :   systeminit.c
**  摘要 :     该文件执行宏初始化。
**  APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备 :     uPD78F0537
**
**  编译器 :   NEC/CC78K0
**
*****************************************************************************
*/
/*
*****************************************************************************
**  包含文件
*****************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
```

```
#include "port.h"
#include "timer.h"
#include "watchdogtimer.h"
#include "lvi.h"
/*
***********************************************************************************
**  宏定义
***********************************************************************************
*/

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     初始化每个宏
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void SystemInit( void )
{
       /* 时钟生成器初始化 */
       Clock_Init();
       /* 端口初始化 */
       PORT_Init();
       /* WDT 初始化 */
       WDT_Init();
       /* TM00 初始化 */
       TM00_Init();
       /* TMH1 初始化 */
       TMH1_Init();
       /* LVI 初始化 */
       LVI_Init();
}

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     硬件设定初始化
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void hdwinit( void )
{
       DI( );
       SystemInit( );
       EI( );
}
```

3.5 System.c

```
/*
************************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名 :  system.c
**  摘要 :    This file implements device driver for System module.
**  APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备 :    uPD78F0537
**
**  编译器 :  NEC/CC78K0
**
************************************************************************************
*/
/*
************************************************************************************
**  包含文件
************************************************************************************
*/
#include "macrodriver.h"
#include "system.h"
/*
************************************************************************************
**  宏定义
************************************************************************************
*/

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     初始化时钟发生器和震荡稳定时间
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void Clock_Init( void )
{
       ClrIORBit(MCM, 0x05);                    /* 用于 CPU 的高速震荡操作 */

       SetIORBit(MCM, 0x01);                    /* 外围硬件时钟:frh */
       ClrIORBit(OSCCTL, 0x10);
       SetIORBit(MOC, 0x80);                    /* 停止 X1 时钟 */
       PCC = CG_CPU_CLOCKSEL;
}
```

3.6 Lvi.h

```
/*
************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名 :  lvi.h
**  摘要 :    该文件执行用于 LVI 模块的设备驱动
**  APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备 :    uPD78F0537
**
**  编译器 :  NEC/CC78K0
**
************************************************************************
*/
#ifndef      _MDLVI_
#define      _MDLVI_
/*
************************************************************************
**  宏定义
************************************************************************
*/
enum LVILevel {
      LVI_Level0, LVI_Level1, LVI_Level2, LVI_Level3,
      LVI_Level4, LVI_Level5, LVI_Level6, LVI_Level7,
      LVI_Level8, LVI_Level9, LVI_Level10, LVI_Level11,
      LVI_Level12, LVI_Level13, LVI_Level14,LVI_Level15
      };

void LVI_Init( void );
void LVI_Start( void );
MD_STATUS LVI_SetLVILevel( enum LVILevel level );
void LVI_Stop( void );
__interrupt void MD_INTLVI( void );

/* 增加示例程序中用于复位模式的 LVI_Start()的专用版本 */
void LVI_Start_R( void );

#endif
```

3.7 Lvi.c

```
/*
************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
```

```
**
**
**   文件名 :   lvi.c
**   摘要 :     该文件执行用于 LVI 模块的设备驱动。
**   APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**   设备 :     uPD78F0537
**
**   编译器 :   NEC/CC78K0
**
*******************************************************************************
*/
/*
*******************************************************************************
** 包含文件
*******************************************************************************
*/
#include "macrodriver.h"
#include "lvi.h"

/*
**-----------------------------------------------------------------------------
**
**   摘要:
**      该函数初始化低电压检测器。
**
**   参数:
**      无
**
**   返回:
**      无
**
**-----------------------------------------------------------------------------
*/
void LVI_Init( void )
{
       USHORT i;

       LVION = 0;                  /* 停止 LVI */
       LVISEL = 0;                 /* 检测供电电压(VDD) */
       LVIMD = 0;                  /* 内部中断模式 */
       LVIPR = 1;                  /* 低优先级 */
       LVIIF = 0;
       LVIS = LVI_Level2;          /* 和 3.91V 比较 */

       LVION = 1;                  /* 允许 LVI */
       for(i=0; i<=100; i++){            /* 等待 10 us */
              NOP();
       }
       /* 为了示范程序，这里不要解除 LVI 中断屏蔽 */
#if 0
       LVIMK = 0;
#endif
}

/*
**-----------------------------------------------------------------------------
**
**   摘要:
**      该函数开始低电压检测器。
**
**   参数:
**      无
**
**   返回:
**      无
**
**-----------------------------------------------------------------------------
```

```
*/
void LVI_Start( void )
{
	USHORT  i;
	LVIMK = 1;
	LVIIF = 0;

	LVION = 1;                  /* 允许 LVI */
	for(i=0; i<=100; i++){           /* 等待 10 us */
		NOP();
	}
	LVIMK = 0;
}

/*
**-------------------------------------------------------------------------------
**
**  摘要:
**     该函数停止低电压检测。
**
**  参数:
**     无
**
**  返回:
**     无
**
**-------------------------------------------------------------------------------
*/
void LVI_Stop( void )
{
	LVION = 0;                  /* 停止 LVI */
	LVIMK = 1;
}

/*
**-------------------------------------------------------------------------------
**
**  摘要:
**     该函数设定检测电平。
**
**  参数:
**     level :       15 个检测电平
**            ( LVI_Level0 - LVI_Level15 )
**
**  返回:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------------
*/
MD_STATUS LVI_SetLVILevel( enum LVILevel level )
{
	if( LVISEL == 1 ){
		return MD_ERROR;
	}
	LVIS = level;
	return MD_OK;
}
```

3.8 Lvi_user.c

```
/*
*******************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
```

```
**
**  Copyright(C) NEC Electronics Corporation 2002 – 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名:    lvi_user.c
**  摘要:      该文件执行用于 LVI 模块的设备驱动。
**  APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备:      uPD78F0537
**
**  编译器:    NEC/CC78K0
**
*******************************************************************************
*/
#pragma      interrupt     INTLVI MD_INTLVI
/*
*******************************************************************************
**  包含文件
*******************************************************************************
*/
#include "macrodriver.h"
#include "lvi.h"
/* add include file for LEDs */
#include "led_0537.h"
/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     INTLVI 中断服务程序。
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTLVI( void )
{
       LVI_Stop();          /* 评比 LVI 操作 */
       led_out_left(0xC7);               /* 左边 LED 显示'L' */
       led_out_right(LED_PAT_1);  /* 右边 LED 显示 1 */
       /* and return */
}

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     该函数为复位模式，通过不清除中断屏蔽 LVIMK 来开始低电压检测器
**
**             (通过 Applilet 生成从 LVI_Start()来修改这个程序)
**
**  参数:
**     无
**
**  返回:
**     无
```

```
**
**-------------------------------------------------------------------------
*/
void LVI_Start_R( void )
{
       USHORT  i;
       LVIMK = 1;
       LVIIF = 0;

       LVION = 1;                  /* 允许 LVI */
       for(i=0; i<=100; i++){            /* 等待 10 us */
              NOP();
       }
/*     LVIMK = 0; */ /* 注释出去 */
}
```

3.9 Port.h

```
/*
*****************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 – 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名 :   port.h
**  摘要 :     该文件执行用于端口模块的设备驱动。
**  APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备 :     uPD78F0537
**
**  编译器 :   NEC/CC78K0
**
*****************************************************************************
*/
#ifndef       _MDPORT_
#define       _MDPORT_
/*
*****************************************************************************
**  宏定义
*****************************************************************************
*/
#define       PORT_PM0      0xff
#define       PORT_PU0      0x0
#define       PORT_P0       0x0
#define       PORT_PM1      0xff
#define       PORT_PU1      0x0
#define       PORT_P1       0x0
#define       PORT_PM2      0xff
#define       PORT_P2       0x0
#define       PORT_PM3      0xff
#define       PORT_PU3      0x6
#define       PORT_P3       0x0
#define       PORT_PM4      0xf0
#define       PORT_PU4      0x0
#define       PORT_P4       0x0
#define       PORT_PM5      0xf0
```

```
#define       PORT_PU5     0x0
#define       PORT_P5      0x0
#define       PORT_PM6     0xff
#define       PORT_P6      0x0
#define       PORT_PM7     0x0
#define       PORT_PU7     0x0
#define       PORT_P7      0x0
#define       PORT_PM12    0xff
#define       PORT_PU12    0x0
#define       PORT_P12     0x0
#define       PORT_P13     0x0
#define       PORT_PM14    0xff
#define       PORT_PU14    0x0
#define       PORT_P14     0x0
#define       PORT_ADPC    0x0

void PORT_Init( void );

#endif
```

3.10 Port.c

```
/*
************************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 – 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名 :   port.c
**  摘要 :     该文件执行用于端口模块的设备驱动。
**  APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备 :     uPD78F0537
**
**  编译器 :   NEC/CC78K0
**
************************************************************************************
*/
/*
************************************************************************************
**  包含文件
************************************************************************************
*/
#include "macrodriver.h"
#include "port.h"
/*
************************************************************************************
**  常数
************************************************************************************
*/

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     该函数初始化 I/O 模块。
**
**  参数:
**     无
```

```
**
**  返回:
**     无
**
**-------------------------------------------------------------------------------
*/
void PORT_Init( void )
{
       /* 初始化端口寄存器 */
       P0 = PORT_P0;
       P1 = PORT_P1;
       P2 = PORT_P2;
       P3 = PORT_P3;
       P4 = PORT_P4;
       P5 = PORT_P5;
       P6 = PORT_P6;
       P7 = PORT_P7;
       P12 = PORT_P12;
       P13 = PORT_P13;
       P14 = PORT_P14;

       /* 初始化上拉电阻选项寄存器 */
       PU0 = PORT_PU0;
       PU1 = PORT_PU1;
       PU3 = PORT_PU3;
       PU4 = PORT_PU4;
       PU5 = PORT_PU5;
       PU7 = PORT_PU7;
       PU12 = PORT_PU12;
       PU14 = PORT_PU14;

       /* 初始化模式寄存器 */
       PM0 = PORT_PM0;
       PM1 = PORT_PM1;
       PM2 = PORT_PM2;
       ADPC = PORT_ADPC;
       PM3 = PORT_PM3;
       PM4 = PORT_PM4;
       PM5 = PORT_PM5;
       PM6 = PORT_PM6;
       PM7 = PORT_PM7;
       PM12 = PORT_PM12;
       PM14 = PORT_PM14;
}
```

3.11 Timer.h

```
/*
*********************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 – 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名 : timer.h
**  摘要 : 该文件执行用于定时器模块的设备驱动
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备 :    uPD78F0537
**
```

```
**  编译器 :  NEC/CC78K0
**
*****************************************************************************
*/

#ifndef _MDTIMER_
#define _MDTIMER_
/*
*****************************************************************************
**  编译器
*****************************************************************************
*/
#define      REGVALUE_MAX  0xff
#define      TM_TM00_CLOCK 0x0
#define      TM_TM00_INTERVALVALUE      0x1f3f
#define      TM_TM00_SQUAREWIDTH 0x1f3f
#define      TM_TM00_PPGCYCLE    0x1f3f
#define      TM_TM00_PPGWIDTH    0x00
#define      TM_TM00_ONESHOTCYCLE0x1f3f
#define      TM_TM00_ONEPULSEDELAY      0x00
#define      TM_TM01_CLOCK 0x0
#define      TM_TM01_INTERVALVALUE      0x00
#define      TM_TM01_SQUAREWIDTH 0x00
#define      TM_TM01_PPGCYCLE    0x00
#define      TM_TM01_PPGWIDTH    0x00
#define      TM_TM01_ONESHOTCYCLE0x00
#define      TM_TM01_ONEPULSEDELAY      0x00
#define      TM_TM50_CLOCK 0x2
#define      TM_TM50_INTERVALVALUE      0x00
#define      TM_TM50_SQUAREWIDTH 0x00
#define      TM_TM50_PWMACTIVEVALUE     0x00
#define      TM_TM51_CLOCK 0x2
#define      TM_TM51_INTERVALVALUE      0x00
#define      TM_TM51_SQUAREWIDTH 0x00
#define      TM_TM51_PWMACTIVEVALUE     0x00
#define      TM_TMH0_CLOCK 0x0
#define      TM_TMH0_INTERVALVALUE      0x00
#define      TM_TMH0_SQUAREWIDTH 0x00
#define      TM_TMH0_PWMCYCLE    0x00
#define      TM_TMH0_PWMDELAY    0x00
#define      TM_TMH1_CLOCK 0x5
#define      TM_TMH1_INTERVALVALUE      0x11
#define      TM_TMH1_SQUAREWIDTH 0x11
#define      TM_TMH1_PWMCYCLE    0x11
#define      TM_TMH1_PWMDELAY    0x00
#define      TM_TMH1_CARRIERDELAY0x11
#define      TM_TMH1_CARRIERWIDTH0x00


/* 定时器 00,01,50,51,H0,H1 配置初始化 */
void TM00_Init(void);
void TMH1_Init(void);

/* 定时期开始 */
void TM00_Start(void);
void TMH1_Start(void);

/* 定时器停止 */
void TM00_Stop(void);
void TMH1_Stop(void);
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
MD_STATUS TMH1_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num);
__interrupt void MD_INTTM000(void);
__interrupt void MD_INTTMH1(void);

/* 增加 g_tmh1_count 的声明 */
extern unsigned char g_tmh1_count;
```

```
#endif       /* _MDTIMER_*/
```

3.12 Timer.c

```
/*
************************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 - 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名: timer.c
**  摘要: 该文件执行用于定时器模块的设备驱动
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备:     uPD78F0537
**
**  编译器:  NEC/CC78K0
**
************************************************************************************
*/

/*
************************************************************************************
**  包含文件
************************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"


/*
************************************************************************************
**  宏定义
************************************************************************************
*/
/*TM00 脉冲宽度测量 */

/*TM01 脉冲宽度测量 */

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     该函数初始化 TM00 模块。
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void TM00_Init( )
{
       TMC00=0x00;
                                                /* 内部计数时钟 */
       PRM00 |= TM_TM00_CLOCK;
       SetIORBit(PR0H, 0x40);                   /* 低优先级 */
```

```
      ClrIORBit(IF0H, 0x40);
      /* TM00 间隔 */
      ClrIORBit(CRC00,0x01);
      CR000 = TM_TM00_INTERVALVALUE;
}
/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     该函数开始 TM00 计数。
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void TM00_Start()
{
      TMC00 = 0x0c;                    /* 内部计数时钟开始 */
      ClrIORBit(MK0H, 0x40);                  /* 允许 INTTM000 */
}

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     该函数停止 TM00 计数器并且清除计数寄存器。
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void TM00_Stop()
{
      TMC00=0x0;
      SetIORBit(MK0H, 0x40);                  /* 停止 INTTM000 */
}

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     该函数改变 TM00 条件。
**
**  参数:
**     USHORT* :    array_reg
**     USHORT :     array_num
**  返回:
**     MD_OK
**     MD_ERROR
**
**-----------------------------------------------------------------------------
*/
MD_STATUS TM00_ChangeTimerCondition(USHORT* array_reg,USHORT array_num)
{
   switch (array_num){
         case 2:
                  CR010=*(array_reg + 1);
         case 1:
                  CR000=*(array_reg + 0);
                  break;
```

```
        default:
                return MD_ERROR;
        }
        return MD_OK;
}

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     该函数初始化 TMH1 模块。
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void TMH1_Init( void )
{
      ClrIORBit(TMHMD1, 0x80);
                  /* countclock=fr/128 */
      SetIORBit(TMHMD1, 0x50);
      SetIORBit(PR0H, 0x08);                    /* 低优先级 */
      ClrIORBit(IF0H, 0x08);
      /* TMH1 间隔定时器 */
      ClrIORBit(TMHMD1, 0x8c);
      CMP01 = TM_TMH1_INTERVALVALUE;
}

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     该函数开始 TMH1 计数器。
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void TMH1_Start( void )
{
      /* TMH1 间隔定时期 */
      SetIORBit(TMHMD1, 0x80);
      ClrIORBit(MK0H, 0x08);                    /* 允许 INTTMH1 */
}

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     该函数可以停止 TMH1 计数器操作。
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void TMH1_Stop( void )
```

```
{
      ClrIORBit(TMHMD1, 0x80);
      SetIORBit(MK0H, 0x08);                    /* 禁止 INTTMH1 */
}

/*
**-------------------------------------------------------------------------
**
**  摘要:
**     该函数可以改变 TMH1 条件。
**
**  参数:
**     UCHAR* :      array_reg
**     UCHAR :       array_num
**  返回:
**     MD_OK
**     MD_ERROR
**
**-------------------------------------------------------------------------
*/
MD_STATUS TMH1_ChangeTimerCondition(UCHAR* array_reg,UCHAR array_num)
{
   switch (array_num){
         case 2:
                  CMP11=*(array_reg + 1);
         case 1:
                  CMP01=*(array_reg + 0);
                  break;
         default:
                  return MD_ERROR;
         }
         return MD_OK;
}
```

3.13 Timer_user.c

```
/*
*****************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 – 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名: timer_user.c
**  描述: 该文件执行用于定时器模块的设备驱动
**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备:     uPD78F0537
**
**  编译器:  NEC/CC78K0
**
*****************************************************************************
*/

#pragma sfr
#pragma interrupt  INTTM000 MD_INTTM000
#pragma interrupt  INTTMH1 MD_INTTMH1
/*
*****************************************************************************
**  包含文件
```

```
************************************************************************************
*/
#include "macrodriver.h"
#include "timer.h"

/* 增加包含按键程序 */
#include "sw_0537.h"
/* 增加包含看门狗定时器程序 */
#include "watchdogtimer.h"

/*
************************************************************************************
**  宏定义
************************************************************************************
*/

/* Timer00, Timer01 脉冲宽度测量 */
/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     TM00 INTTM000 中断服务程序
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTTM000( )
{
       EI();  /* 允许为按键去抖动调用按键 ISR 函数中调用产生的高优先级中断(INTTMH1, INTLVI) */
       sw_isr();
}
/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     TMH1 INTTMH1 中断服务程序
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
/* 通过 TMH1 中断增加全局变量 */
unsigned char g_tmh1_count;

__interrupt void MD_INTTMH1( )
{
       EI();  /* 允许产生高优先级中断(INTLVI) */
       WDT_Restart();
       g_tmh1_count = g_tmh1_count + 1;
}
```

3.14 Watchdogtimer.h

```
/*
************************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 – 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件:     watchdogtimer.h
**  摘要:     该文件执行用于 WDT 模块的设备驱动。
**  APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备:     uPD78F0537
**
**  编译器:   NEC/CC78K0
**
************************************************************************************
*/

#ifndef      _MDWTACHDOGTIMER_
#define      _MDWTACHDOGTIMER_
/*
************************************************************************************
**  宏定义
************************************************************************************
*/
void WDT_Init( void );
void WDT_Start( void );
void WDT_Restart( void );

#endif
```

3.15 Watchdogtimer.c

```
/*
************************************************************************************
**
**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
**  的设备驱动通过 Applilet 生成。
**
**  Copyright(C) NEC Electronics Corporation 2002 – 2005
**  NEC Electronics Corporation 版权所有。
**
**  使用该程序是用户自己的责任。
**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
**
**
**  文件名:   watchdogtimer.c
**  摘要:     该文件执行用于 WDT 模块的设备驱动。
**  APIlib :  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
**
**  设备:     uPD78F0537
**
**  编译器:   NEC/CC78K0
**
************************************************************************************
*/

/*
************************************************************************************
**  包含文件
```

```
*****************************************************************************
*/
#include "macrodriver.h"
#include "watchdogtimer.h"
/*
*****************************************************************************
**  宏定义
*****************************************************************************
*/

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     该函数初始化看门狗定时器模块。
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void WDT_Init( void )
{
       /* 溢出时间: 2^14/frl (62.06ms) */
       /* 开窗周期: 100% (溢出时间之前) */
}

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     这个函数开始看门狗定时器。
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
void WDT_Start( void )
{
       WDTE = 0xac;
}

/*
**-----------------------------------------------------------------------------
**
**  摘要:
**     该函数清除并再次开始看门狗定时器。
**
**  参数:
**     无
**
**  返回:
**     无
**
**-----------------------------------------------------------------------------
*/
```

```
void WDT_Restart( void )
{
      WDTE = 0xac;
}
```

3.16 Led_0537.h

```
/* led_0537.h                                                                    */
/*     用于 LED 显示的 M-78F0537 CPU 板的头文件      */
/* 版本 1.1    01-13-2006                                                        */

#ifndef _LED_0537_H
#define _LED_0537_H

/*********************************************************************/
/* 详细定义                                                          */
/*********************************************************************/

/* 十进制和十六进制 LED 图形，符号 */
/* 单独的位,           ---A---      */
/* 0=开 1=关          |       |     */
/* 位 0 = 段 A        F       B     */
/* 位 1 = 段 B        |       |     */
/* 位 2 = 段 C         ---G---      */
/* 位 3 = 段 D        |       |     */
/* 位 4 = 段 E        E       C     */
/* 位 5 = 段 F        |       |     */
/* 位 6 = 段 G         ---D--- DP   */
/* 位 7 = 十进制小数点                      */

#define LED_PAT_0   0xC0
#define LED_PAT_1   0xF9
#define LED_PAT_2   0xA4
#define LED_PAT_3   0xB0
#define LED_PAT_4   0x99
#define LED_PAT_5   0x92
#define LED_PAT_6   0x82
#define LED_PAT_7   0xF8
#define LED_PAT_8   0x80
#define LED_PAT_9   0x98
#define LED_PAT_A   0x88
#define LED_PAT_B   0x83
#define LED_PAT_C   0xC6
#define LED_PAT_D   0xA1
#define LED_PAT_E   0x86
#define LED_PAT_F   0x8E
#define LED_PAT_BLANK      0xFF
#define LED_PAT_DP         0x7F
#define LED_PAT_DASH0xBF
#define LED_PAT_ULINE      0xF7
#define LED_PAT_OLINE      0xFE
#define LED_PAT_EQUAL      0xB7

/*********************************************************************/
/* 出口函数                                                          */
/*********************************************************************/
extern void led_init(void);                                  /* 初始化用于 LED 输出的端口    */
extern void led_out_right(unsigned char val);  /* 右边 LED 的输出值    */
extern void led_out_left(unsigned char val);   /* 左边 LED 的输出值          */
extern void led_dig_right(unsigned char num);  /* 右边 LED 显示数字    */
extern void led_dig_left(unsigned char num);   /* 左边 LED 显示数字    */
extern void led_dig(unsigned char num);              /* 显示十六进制数字           */
extern void led_dig_bcd(unsigned char bcdnum); /* 显示 BCD 码数字            */

#endif /* _LED_0537_H */
```

3.17 Led_0537.c

```
/*     led_0537.c - 用于 LED 显示的程序                                */
/*            用于 M-Station 基础板上的 M-78F0537 CPU 板   */
/*  版本:     1.1    01-13-2006                                        */

/*            P70-P77 = 输出到右边数字 (LED2)                  */
/*            P40-P43 = 输出到左边数字  (LED1) 位 0-3    */
/*            P50-P53 = 输出到左边数字  (LED1) 位 4-7    */

/*     要连接端口到 M-Station 1.1，确保连接下列 ROW1 和 ROW2 之间的跳线。
       要连接端口到 M-Station 2，确保插入缺省的 SBxx 连接。
       Port   LED          M-Station 1.1 M-Station 2.2
       ----   ---          ------------- -------------
       P70    2-A          R1.25 - R2.25 SB27
       P71    2-B          R1.26 - R2.26 SB28
       P72    2-C          R1.27 - R2.27 SB29
       P73    2-D          R1.28 - R2.28 SB30
       P74    2-E          R1.29 - R2.29 SB31
       P75    2-F          R1.30 - R2.30 SB32
       P76    2-G          R1.31 - R2.31 SB33
       P77    2-DP         R1.32 - R2.32 SB34

       P40    1-A          R1.17 - R2.17 SB35
       P41    1-B          R1.18 - R2.18 SB36
       P42    1-C          R1.19 - R2.19 SB37
       P43    1-D          R1.20 - R2.20 SB38

       P50    1-E          R1.21 - R2.21 SB39
       P51    1-F          R1.22 - R2.22 SB40
       P52    1-G          R1.23 - R2.23 SB41
       P53    1-DP         R1.24 - R2.24 SB42
*/

/* 注: 在 M-Station 基本 V1.0 保护类型下，P40-P53 定向为 ROW4.1-8，       */
/*  需要用导线连接 ROW2.17-24 到 LED1 驱动    */

/* 需要声明在 C 中访问 SFR's   */
#pragma sfr

#include "led_0537.h"

/* 七段数字为组合表 */
static unsigned char dig_tab[] = {
   LED_PAT_0, /* 0 */
   LED_PAT_1, /* 1 */
   LED_PAT_2, /* 2 */
   LED_PAT_3, /* 3 */
   LED_PAT_4, /* 4 */
   LED_PAT_5, /* 5 */
   LED_PAT_6, /* 6 */
   LED_PAT_7, /* 7 */
   LED_PAT_8, /* 8 */
   LED_PAT_9,        /* 9 */
   LED_PAT_A, /* A */
   LED_PAT_B, /* B */
   LED_PAT_C, /* C */
   LED_PAT_D, /* D */
   LED_PAT_E, /* E */
   LED_PAT_F  /* F */
};

/*     void led_init(void) */
/*            设定用于显示 LED 的端口 */
void led_init(void)
{
```

```
#if 0  /* 通过 Applilet 在 Port_Init()中端口初始化 */
       PM7 = 0x00;          /* 设定全部端口 7 输出 */
       PM4 = 0x00;          /* 设定全部端口 4 输出 */
       PM5 = 0x00;          /* 设定全部端口 5 输出 */
#endif
}

/* void led_out_right(unsigned char val) */
/*            输出原始数据到右边 LED */
void led_out_right(unsigned char val)
{
       P7 = val;
}

/* void led_out_left(unsigned char val) */
/*            输出原始数据到左边 LED */
void led_out_left(unsigned char val)
{
       P4 = val & 0x0F;
       P5 = (val >> 4) & 0x0F;
}

/* void led_dig_right(unsigned char num) */
/*     在右边 LED 显示数字 */
void led_dig_right(unsigned char num)
{
       if (num > 0x0F) {
              led_out_right(LED_PAT_BLANK);
              return;
       }
       led_out_right(dig_tab[num]);
}

/* void led_dig_left(unsigned char num) */
/*     在左边 LED 显示数字 */
void led_dig_left(unsigned char num)
{
       if (num > 0x0F) {
              led_out_left(LED_PAT_BLANK);
              return;
       }
       led_out_left(dig_tab[num]);
}

/* void led_dig(unsigned char num) */
/*            显示数字为十六进制数字 */
/*            num – 显示的数字 */
/*                   位 0-3 右边数字 */
/*                   位 4-7 左边数字 */
void led_dig(unsigned char num)
{
       led_out_right(dig_tab[num & 0x0F]);
       led_out_left(dig_tab[(num >> 4) & 0x0F]);
}

/* void led_dig_bcd(unsigned char bcdnum) */
/*            显示 BCD 编码的两位数字 */
/*            bcdnum – 在 BCD 下显示的数字 */
/*                   0 - 9      在右边显示数字，左边为空 */
/*                   10 - 99    显示两位数字 */
/*                   100 - 255 显示空 */
void led_dig_bcd(unsigned char bcdnum)
{
unsigned char tens_dig;

       if (bcdnum > 99) {
              led_out_right(LED_PAT_BLANK);     /* 两个数字同时显示空 */
              led_out_left(LED_PAT_BLANK);
```

```
            return;
        }

        if (bcdnum < 10) {
            led_out_right(dig_tab[bcdnum]);  /* 至显示右边 LED */
            led_out_left(LED_PAT_BLANK);            /* 左边 LED 空 */
            return;
        }

        /* 10 <= bcdnum <= 99 */
        tens_dig = 0;
        do {                        /* 减 10 来计算十位和余数 */
            bcdnum -= 10;
            tens_dig++;
        } while (bcdnum >= 10);
        /* 现在 tens_dig 放入十位 */
        /* bcdnum 放入余数 */
        led_out_right(dig_tab[bcdnum]);
        led_out_left(dig_tab[tens_dig]);
}
```

3.18 Sw_0537.h

```
/* sw_0537.h */
/*     用于读取按键基础板的 M-78F0537 CPU 板的头文件 */
/*  版本:     1.1    01-13-2006    */

#ifndef _SW_0537_H
#define _SW_0537_H

/*********************************************************************/
/* 详细定义                                                          */
/*********************************************************************/

/* 用于案件输入的符号定义 */
/* SW2 = 左边按键 = P31 */
/* SW3 = 右边按键 = P32 */
/*                                                    P32          P31 */
#define SW_LU_RU    0x06   /* 左边抬起，右边抬起        1            1      */
#define SW_LD_RU    0x04   /* 左边按下，右边抬起         1            0      */
#define SW_LU_RD    0x02   /* 左边抬起，右边按下         0            1      */
#define SW_LD_RD    0x00   /* 左边按下，右边按下         0            0      */

#define       SW_DEF_DEB_COUNT    8      /* 缺省去抖动计数器    */

/*********************************************************************/
/* 出口函数                                                          */
/*********************************************************************/
extern void sw_init(void);              /* 初始化用于按键输入的端口和变量 */
extern unsigned char sw_chk(void);      /* 获得去未抖动按键输入 */
extern unsigned char sw_get(void);      /* 获得未抖动按键输入 */
extern void sw_set_debounce(unsigned char count);     /* 设定去抖动计数 */
extern void sw_isr(void);               /* 去抖动函数，通过定时器 ISR 调用 */

#endif /* _SW_0537_H */
```

3.19 Sw_0537.c

```
/*     sw_0537.c – 用于按键输入的程序                              */
/*            用于 M-Station 基础板上的 M-78F0537 CPU 板    */
/*  版本:     1.1    01-13-2006                                    */

/*            P31 = 为左边按键输入  (SW2)                         */
/*            P32 = 为右边按键输入 (SW3)                          */
```

```
/*     要连接端口到 M-Station 1.1，确保连接下列 ROW1 和 ROW2 之间的跳线。
       要连接端口到 M-Station 2，确保插入缺省的 SBxx 连接。

       Port   按键          M-Station 1.1 M-Station 2.2
       ----   ---           ---------------------------
       P31    SW2           R1.5 - R2.5         SB7
       P32    SW3           R1.6 - R2.6         SB8
*/

/* 需要声明在 C 中访问 SFR's   */
#pragma sfr

#include "sw_0537.h"

/* 用于按键的本地变量 */
static unsigned char sw_last;           /* 上一次去抖动按键值 */
static unsigned char sw_new;            /* 去抖动的新值 */
static unsigned char sw_deb_value;      /* 去抖动计数器的值 */
static unsigned char sw_deb_count;      /* 去抖动计数器 */

/*     void sw_init(void) */
/*           设置端口用于按键输入 */
void sw_init(void)
{
#if 0  /* 通过 Applilet 在 Port_Init()中初始化 */
       /* 设定 P31 和 P32 为输入 */
       PM3.1 = 1;
       PM3.2 = 1;
       /* 设定 P31 和 P32 的上拉电阻 */
       PU3.1 = 1;
       PU3.2 = 1;
#endif
       /* 设定静态变量 */
       sw_last = SW_LU_RU; /* 缺省是右边抬起，左边抬起(没有按键按下) */
       sw_deb_value = SW_DEF_DEB_COUNT;  /* 设定去抖动计数器的缺省值 */
       sw_deb_count = SW_DEF_DEB_COUNT;  /* 设定计数器为最大 */
}

/* unsigned char sw_chk(void) */
/*     从按键返回输入，无去抖动 */
unsigned char sw_chk(void)
{
       return P3 & 0x06;
}

/* void sw_set_debounce(unsigned char count) */
/*     设定去抖动计数器值 */
void sw_set_debounce(unsigned char count)
{
       sw_deb_value = count;      /* 设定新去抖动计数器值 */
       sw_deb_count = count;      /* 设定计数器为最大 */
}

/* unsigned char sw_get(void) */
/*     返回去抖动按键输入 */

unsigned char sw_get(void)
{
       return sw_last;
}
```

```
/* void sw_isr( void ) */
/* 该函数通过周期定时器中断来调用，在一个新值查询和去抖动按键 sw_deb_value 之后，更新 sw_last  */
void sw_isr( void )
{
unsigned char val;

       val = sw_chk();             /* 获得当前值 */
       /* 如果和以前相同，没有改变；复位去抖动并且返回 */
       if (val == sw_last) {
              sw_deb_count = sw_deb_value;      /* 复位去抖动计数器为最大 */
              return;
       }

       /* val != sw_last, 这是一个新的输入 */
       /* 如果和上一个新的不相同，设定新的，复位去抖动计数器并且返回 */
       if (val != sw_new) {
              sw_new = val;
              sw_deb_count = sw_deb_value;
              return;
       }

       /* val != sw_last, val == sw_new */
       /* 去抖动计数器减一 */
       sw_deb_count--;

       /* 如果我们减到零，我们为去抖动计数次数发送相同 sw_new，现在是去抖动按键值 */
       if (sw_deb_count == 0) {
              sw_last = val;
              sw_deb_count = sw_deb_value;
              return;
       }

       /* 如果依然去抖动，返回 */
       return;
}
```

3.20 Option.inc

注: 该版本是为 1.59V POC 模式设计。

```
;*****************************************************************************
;**
;**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
;**  的设备驱动通过 Applilet 生成。
;**
;**  Copyright(C) NEC Electronics Corporation 2002 – 2005
;**  NEC Electronics Corporation 版权所有。
;**
;**  使用该程序是用户自己的责任。
;**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
;**
;**
;**  文件名: option.asm
;**  摘要: 该文件执行选项字节/安全 ID 设定。
;**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;**  设备:    uPD78F0537
;**
;**  编译器:  NEC/CC78K0
;**
;*****************************************************************************

;
```

```
;*******************************************************************************
;**  宏定义
;*******************************************************************************
;
OPTION_BYTE   EQU    00H
POC81  EQU    00H
POC82  EQU    00H
POC83  EQU    00H
CG_ONCHIP     EQU    02H
CG_SECURITY0  EQU    0ffH
CG_SECURITY1  EQU    0ffH
CG_SECURITY2  EQU    0ffH
CG_SECURITY3  EQU    0ffH
CG_SECURITY4  EQU    0ffH
CG_SECURITY5  EQU    0ffH
CG_SECURITY6  EQU    0ffH
CG_SECURITY7  EQU    0ffH
CG_SECURITY8  EQU    0ffH
CG_SECURITY9  EQU    0ffH
OPTION_BYTE_WDT     EQU    078H
```

3.21 Option.asm

注: 该版本是为 1.59V POC 模式设计。

```
;*******************************************************************************
;**
;**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
;**  的设备驱动通过 Applilet 生成。
;**
;**  Copyright(C) NEC Electronics Corporation 2002 – 2005
;**  NEC Electronics Corporation 版权所有。
;**
;**  使用该程序是用户自己的责任。
;**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
;**
;**
;**  文件名: option.asm
;**  摘要: 该文件执行选项字节/安全 ID 设定。
;**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;**  设备:    uPD78F0537
;**
;**  编译器:  NEC/CC78K0
;**
;*******************************************************************************

;
;*******************************************************************************
;**  包含文件
;*******************************************************************************
$ INCLUDE (option.inc)
       OPT_SET CSEG AT 80H
OPTION:       DB     OPTION_BYTE + OPTION_BYTE_WDT;
       DB     POC81
       DB     POC82
       DB     POC83
       ONC_SET CSEG AT 84H
ONCHIP:       DB     CG_ONCHIP

       CSEG SECUR_ID
SECURITY0:    DB     CG_SECURITY0
```

```
SECURITY1:    DB     CG_SECURITY1
SECURITY2:    DB     CG_SECURITY2
SECURITY3:    DB     CG_SECURITY3
SECURITY4:    DB     CG_SECURITY4
SECURITY5:    DB     CG_SECURITY5
SECURITY6:    DB     CG_SECURITY6
SECURITY7:    DB     CG_SECURITY7
SECURITY8:    DB     CG_SECURITY8
SECURITY9:    DB     CG_SECURITY9
END
```

3.22 Option_POC27.inc

注: 该版本是为 2.7V1.59V POC 模式设计。

```
;*******************************************************************************
;**
;**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
;**  的设备驱动通过 Applilet 生成。
;**
;**  Copyright(C) NEC Electronics Corporation 2002 – 2005
;**  NEC Electronics Corporation 版权所有。
;**
;**  使用该程序是用户自己的责任。
;**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
;**
;**
;**  文件名: option.asm
;**  摘要: 该文件执行选项字节/安全 ID 设定。
;**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;**  设备:    uPD78F0537
;**
;**  编译器:  NEC/CC78K0
;**
;*******************************************************************************

;
;*******************************************************************************
;**  宏定义
;*******************************************************************************
;
OPTION_BYTE   EQU    00H
; original POC81 byte as generated by Applilet
; had POC81.0 = 0 (POCMODE) to select 1.59V POC mode
; for SAFETY_POC27 project, changed value of POC81 to 01H
; so POC81.0 = 1 (POCMODE) to select 2.7V/1.59V POC mode
;POC81 EQU    00H
POC81  EQU    01H
POC82  EQU    01H
POC83  EQU    00H
CG_ONCHIP     EQU    02H
CG_SECURITY0  EQU    0ffH
CG_SECURITY1  EQU    0ffH
CG_SECURITY2  EQU    0ffH
CG_SECURITY3  EQU    0ffH
CG_SECURITY4  EQU    0ffH
CG_SECURITY5  EQU    0ffH
CG_SECURITY6  EQU    0ffH
CG_SECURITY7  EQU    0ffH
CG_SECURITY8  EQU    0ffH
CG_SECURITY9  EQU    0ffH
```

```
OPTION_BYTE_WDT     EQU    078H
```

3.23 Option_POC27.asm

注: 该版本是为 2.7V1.59V POC 模式设计。

```
;*******************************************************************************
;**
;**  这个用于 78K0/KB2, 78K0/KC2, 78K0/KD2, 78K0/KE2 和 78K0/KF2 8 位单片微控制器
;**  的设备驱动通过 Applilet 生成。
;**
;**  Copyright(C) NEC Electronics Corporation 2002 – 2005
;**  NEC Electronics Corporation 版权所有。
;**
;**  使用该程序是用户自己的责任。
;**  NEC Electronics Corporation 不对用户或者第三方使用该文件发生的任何损失负责。
;**
;**
;**  文件名: option.asm
;**  摘要: 该文件执行选项字节/安全 ID 设定。
;**  APIlib:  NEC78K0KX2.lib V1.01 [09 Aug. 2005]
;**
;**  设备:    uPD78F0537
;**
;**  编译器:  NEC/CC78K0
;**
;*******************************************************************************

;
;*******************************************************************************
;**  包含文件
;*******************************************************************************
; 为项目 SAFETY_POC27 project 通过 Applilet 作为最初改变 option.asm
;  为 option_POC27.asm，包括 option_POC27.inc
;$ INCLUDE (option.inc)
$ INCLUDE (option_POC27.inc)
       OPT_SET CSEG AT 80H
OPTION:      DB     OPTION_BYTE + OPTION_BYTE_WDT;
       DB     POC81
       DB     POC82
       DB     POC83
       ONC_SET CSEG AT 84H
ONCHIP:      DB     CG_ONCHIP

       CSEG SECUR_ID
SECURITY0:   DB     CG_SECURITY0
SECURITY1:   DB     CG_SECURITY1
SECURITY2:   DB     CG_SECURITY2
SECURITY3:   DB     CG_SECURITY3
SECURITY4:   DB     CG_SECURITY4
SECURITY5:   DB     CG_SECURITY5
SECURITY6:   DB     CG_SECURITY6
SECURITY7:   DB     CG_SECURITY7
SECURITY8:   DB     CG_SECURITY8
SECURITY9:   DB     CG_SECURITY9
END
```